Universitat de Girona
Escola Politècnica Superior

Projecte/Treball Final de Carrera

Estudi: Enginyeria Informàtica. Pla 1997

Títol: **Optimització de la tècnica dels Miralls Màgics a partir de mesures de Teoria de la Informació. Aplicació a la visualització 3D de dades mèdiques.**

Document: Memòria

Alumne: Marc Ruiz Altisent

Director/Tutor: Imma Boada Oliveras
Departament: Informàtica i Matemàtica Aplicada
Àrea: LSI

Convocatòria (mes/any): Setembre 2006

Índex de continguts

1 Introducció i objectius

1.1 Antecedents

La **visualització científica** és una àrea de la informàtica gràfica que té com a objectiu principal **fer comprensibles grans volums d'informació a partir d'interpretacions gràfiques**. La visualització científica estudia i defineix algorismes i estructures de dades que permeten fer comprensibles aquests conjunts de dades a través d'imatges.

Els camps d'aplicació de la visualització científica són molts i diversos: geologia, medicina, química, meteorologia, ciències de la terra, etc. Aquesta diversitat d'àrees fa que els algorismes i les tècniques que es desenvolupin depenguin directament de les característiques de les dades de les quals es disposa inicialment. Podem dir per tant que la majoria de tècniques de visualització científica s'han desenvolupat per a resoldre problemes concrets.

Actualment, el diagnòstic mitjançant la imatge mèdica s'ha convertit en una eina fonamental en la pràctica clínica diària, ja que permet, entre d'altres coses, reconstruir a partir d'un conjunt d'imatges 2D obtingudes a partir d'aparells de captació qualsevol part de l'organisme d'un pacient i representar-lo en un model 3D. Aquest model 3D s'anomena **model de vòxels** sobre ell poden realitzar-se diferents operacions que faciliten el diagnòstic i la presa de decisions als especialistes. Entre aquestes operacions hi ha visualitzacions per pantalla, càlcul de mesures, simulacions d'inserció de pròtesis, etc.

Malgrat els avantatges que proporciona la visualització científica tridimensional a l'hora d'ajudar a interpretar les dades, la majoria de centres hospitalaris visualitzen les dades com a conjunts d'imatges 2D. D'altra banda, una de les línies d'investigació del Grup de Gràfics de Girona (GGG) de la Universitat de Girona (UdG) és l'aplicació de la visualització científica en medicina. Per aquesta raó s'ha establert un conveni de col·laboració entre el GGG i el grup de neuro-radiologia de l'Institut de Diagnòstic per la Imatge (IDI) de l'Hospital Universitari Dr. Josep Trueta de Girona. En el marc d'aquest conveni la primera fita que s'ha marcat és la de desenvolupar una plataforma de visualització i manipulació de dades mèdiques que incorpori les tècniques bàsiques de visualització científica. Aquesta plataforma pretén complementar la visualització bidimensional (2D) majoritària en l'entorn mèdic amb una visualització tridimensional (3D) que permeti inspeccionar la informació captada del pacient de forma més eficient i facilitant-ne el seu diagnòstic. Aquesta plataforma, anomenada **StarViewer**, inclourà algorismes de registre de dades (combinació de models), segmentació i visualització.

1.2 Objectius

La utilitat d'un model de vòxels depèn del punt de vista des del qual es visualitzi. Per aconseguir la màxima utilitat cal poder-lo veure des del **punt de vista ideal**, és a dir, el que aporta més informació.

D'altra banda, hi ha una tècnica de visualització anomenada **Miralls Màgics** que permet veure un model de vòxels des de diferents punts de vista alhora i amb una visualització diferent a cada mirall.

En aquest projecte implementarem la tècnica dels Miralls Màgics i un algorisme que ajudarà a determinar el punt de vista ideal per visualitzar un model de vòxels així com també els punts de vista ideals per als miralls per tal d'aconseguir el màxim d'informació possible del model de vòxels. Aquest algorisme es basa en la teoria de la informació per saber quina és la millor visualització. L'algorisme també permetrà trobar una assignació de colors que sigui útil a l'hora de visualitzar el model de vòxels. El model de vòxels codifica valors de propietats capturades d'un pacient i per poder visualitzar-les en la pantalla cal que a cada propietat li assignem un color determinat.

Cal remarcar que l'algorisme que implementarem no trobarà els millors punts de vista sinó que proporcionarà informació que pot ajudar a l'usuari a decidir quins són els millors. Així mateix, tampoc no donarà l'assignació de colors òptima, sinó una que permeti veure cada regió del model d'un color diferent. La idea és desenvolupar el marc de treball que permeti obtenir de forma automàtica les dades necessàries per poder realitzar un estudi a partir del qual es podrà proposar una tècnica per obtenir la millor vista.

Per poder assolir aquests objectius caldrà:

- Implementar la tècnica bàsica dels Miralls Màgics o *Magic Mirrors*. Aquesta tècnica, explicada de forma senzilla, consisteix en fer la visualització d'un model de vòxels al centre de la pantalla i situar al seu voltant uns plans semblants a uns miralls on hi ha la imatge del mateix model vist des de la posició del mirall. Els miralls faciliten la interpretació de qualsevol model de vòxels ja que permeten a l'usuari visualitzar-los des de diferents punts de vista de forma simultània.

- Estendre la tècnica dels Miralls Màgics per tal de suportar models fusionats. Un model fusionat és un model de vòxels que té més d'un valor de propietat a cada vòxel. Com que els models fusionats tenen més d'una propietat a cada vòxel els Miralls Màgics han de ser capaços de visualitzar diferents propietats en cadascun dels miralls.

- Permetre qualsevol nombre de miralls en qualsevol posició. L'usuari ha de poder triar el nombre de miralls i alterar les seves posicions.

- Permetre qualsevol combinació dels paràmetres de visualització. L'usuari ha de poder decidir quines propietats vol veure en cada mirall i de quina manera (de quin color).

- Implementar una interfícies d'usuari per poder definir i modificar fàcilment les assignacions de colors als valors de propietat.

- Definir i implementar una tècnica basada en mesures de Teoria de la Informació que ajudi a determinar com s'han de col·locar els Miralls Màgics per aconseguir representar el màxim d'informació en pantalla.

A banda d'aquests objectius concrets, també hi ha uns objectius més generals que caldrà satisfer.

El primer de tots és que la implementació d'aquestes tècniques s'ha d'integrar a la plataforma StarViewer. Això condicionarà el disseny de manera que aquest haurà de ser modular i haurà de seguir una estructura de classes determinada.

També cal que l'aplicació tingui una interfície gràfica d'usuari que permeti accedir a totes les funcionalitats de l'aplicació. Aquesta interfície ha de ser senzilla d'utilitzar i tant intuïtiva com sigui possible. A més a més ha d'estar correctament integrada a la interfície de l'StarViewer.

Altres objectius són que l'aplicació ha de ser relativament ràpida per permetre una alta interactivitat, però sense renunciar a les funcionalitats més importants.

Finalment, cal que l'aplicació es construeixi fent servir eines de domini públic, per tal que la seva modificació o ampliació no impliqui costos elevats. Aquestes eines estan determinades per la plataforma i són les següents: llenguatge de programació C++, biblioteques Qt per construir la interfície gràfica, biblioteques ITK per al tractament d'imatges i biblioteques VTK per a la visualitza d'imatges. A més a més, l'entorn de treball serà un sistema operatiu GNU/Linux, amb el compilador GCC i l'entorn de desenvolupament KDevelop.

1.3 Metodologia emprada

La programació de l'aplicació realitzada en aquest projecte segueix el paradigma d'Orientació a Objectes (OO), per tant cal seguir una metodologia adaptada a aquest paradigma de programació.

Els diagrames construïts en les fases d'anàlisi i disseny de l'aplicació segueixen l'estàndard UML (*Unified Modeling Language*). Es pot trobar més informació sobre aquesta especificació a [1] i [2].

La metodologia utilitzada per realitzar el projecte ha estat *eXtreme Programming* (XP). Aquesta metodologia dóna més flexibilitat que altres metodologies a l'hora de modificar el disseny de l'aplicació durant la implementació. Com que en aquest projecte el disseny depèn de detalls d'implementació (per les possibilitats i limitacions de les biblioteques utilitzades) hem considerat que aquesta era la metodologia més adequada, ja que la millor manera de conèixer les biblioteques és fent proves i algunes d'aquestes proves no es poden fer sense haver implementat una part de l'aplicació. Això implica dissenyar i implementar alhora, que és la idea bàsica de la metodologia XP. Es pot trobar més informació sobre aquesta metodologia a [3].

1.4 Planificació

El treball realitzat per poder assolir els objectius marcats en el projecte el podem agrupar en dues fases diferents.

- Una primera fase en la qual s'ha realitzat l'estudi teòric necessari per poder adquirir els coneixements requerits per poder començar el disseny i la implementació.

- Una segona fase en la qual s'han dissenyat, implementat i integrat en la plataforma StarViewer tots els mòduls necessaris per poder assolir els objectius.

Tenint en compte aquesta planificació hem dividit aquest document en els capítols següents:

1. **Introducció i objectius:** Una breu introducció al projecte i els objectius.

2. **Fonaments teòrics:** Conceptes teòrics que ha calgut aprendre abans de començar el disseny.

3. **Fonaments pràctics:** Informació sobre les biblioteques principals utilitzades: Qt, VTK, ITK.

4. **Anàlisi i visió general de l'aplicació:** Anàlisi de requeriments de l'aplicació que cal dissenyar, explicant cada cas d'ús, i una introducció als mòduls que la compondran.

5. **Disseny de l'aplicació:** Explicació detalla del disseny de cada mòdul.

6. **Implementació:** Explicació de l'aplicació ja construïda des del punt de vista del que pot fer l'usuari.

7. **Avaluació dels resultats:** Resultats obtinguts amb diferents proves, avaluació del rendiment i valoració dels resultats.

8. **Millores i ampliacions:** Millores que es podrien fer sobre el disseny o la implementació de l'aplicació.

9. **Conclusions:** Conclusions obtingudes després de realitzar aquest projecte i la planificació seguida.

   Referències: Bibliografia i llocs web consultats.

   Manual d'usuari: Com es fa servir l'aplicació.

2 Fonaments teòrics

En aquest capítol exposarem els fonaments teòrics que han servit de base per a la realització del projecte. Són els conceptes que calia conèixer i entendre per dur a terme el projecte i no tenen res a veure amb els detalls de la implementació.

2.1 Introducció

La visualització científica és l'àrea de la informàtica que estudia i defineix els algorismes i estructures de dades que permeten interpretar conjunts de dades a través d'imatges. El principal objectiu d'aquesta àrea de la informàtica gràfica és fer comprensibles grans volums d'informació a partir d'interpretacions gràfiques. Una de les principals àrees d'aplicació d'aquest tipus de visualització informàtica és la medicina.

Els avenços tecnològics que s'han produït en aquests darrers anys han transformat considerablement les característiques del tractament de les imatges mèdiques. L'aparició de dispositius de captura sofisticats com són els equips de tomografia computeritzada (CT), ressonància magnètica (MRI), etc. han marcat un canvi revolucionari sobre les pràctiques de diagnòstic en permetre la reconstrucció de talls interns del cos humà amb alta precisió. D'altra banda les millores en el camp de la informàtica gràfica i la reducció de costos dels equips personals ha permès desenvolupar aplicacions que abans estaven restringides a grans equipaments. D'aquesta forma és factible actualment la implementació d'entorns de visualització i anàlisi d'imatges tridimensionals generades a partir de dispositius de captació, proporcionant eines de caràcter no invasiu per a l'assistència dels professionals en diferents tipus de procediments mèdics.

Malgrat tots aquests avenços hi ha molts problemes que encara queden per resoldre. En aquest projecte ens centrarem en alguns d'aquests problemes. Presentarem la problemàtica i proposarem com solucionar-la. En particular ens centrarem en la **visualització de models fusionats**, la **selecció del punt de vista** i l'**assignació de colors**.

Visualització de models fusionats

En general la informació que s'obté del pacient a través dels diferents dispositius de captació és informació complementària, això ha fet que la possibilitat d'integrar tota aquesta informació en un model de representació únic hagi esdevingut una eina de gran valor pels especialistes. El procés que s'aplica per poder integrar tota la informació en un únic model és el que es coneix com a procés de registre.

El registre de models no és senzill. Hem de tenir en compte que **els conjunts de dades proporcionats pels diferents dispositius de captació poden tenir resolucions diferents**. Cada volum de dades tindrà la seva pròpia resolució, depenent de la prova que s'hagi fet i la màquina que s'hagi fet servir. Per exemple, un fMRI genera imatges de menys resolució que un MRI. D'altra banda cal tenir en compte que **l'extensió espacial dels conjunts de dades també pot ser diferent**. Podem trobar-nos per exemple, que en un model hi ha una exploració de tot el cap i en l'altre només una part. A la Figura 2.1 hi ha un diagrama amb 2 dispositius de captació diferents i les imatges que generen. Com es pot veure, la mida de les imatges és diferent, i

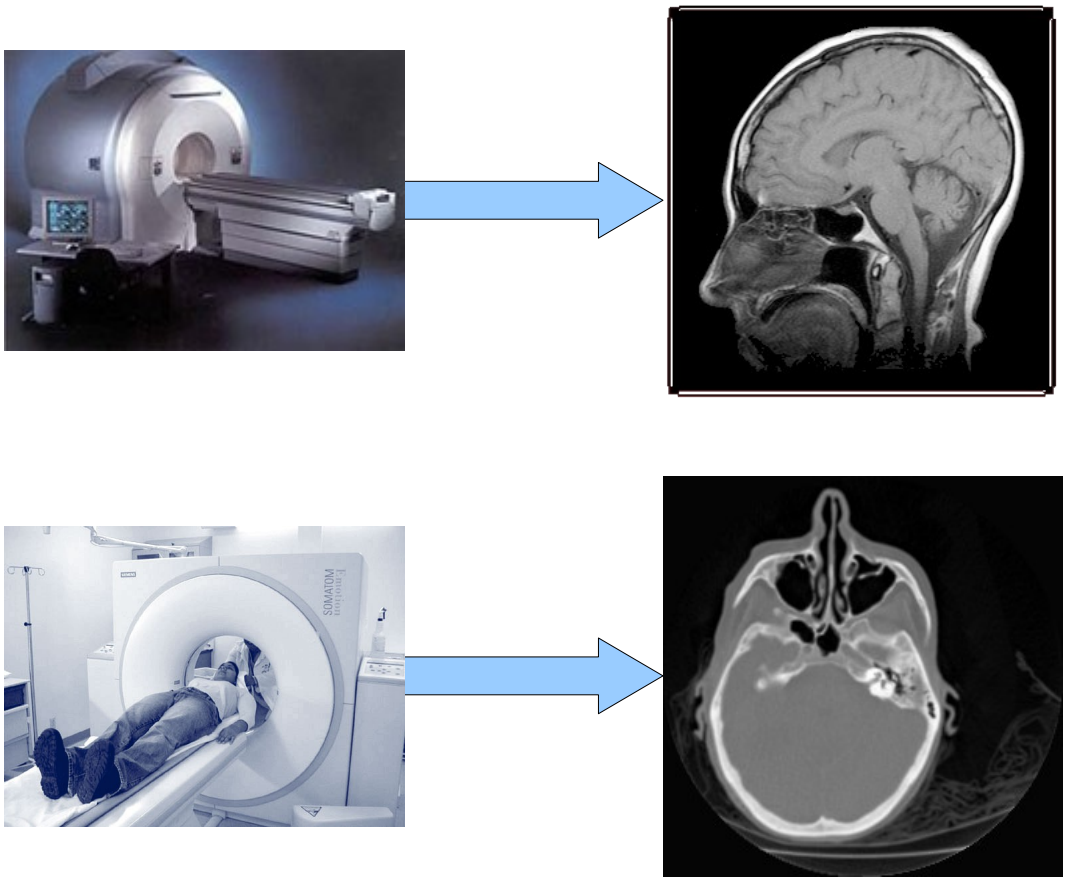


Figura 2.1: Exemples de dispositius de captació i les imatges obtingudes. A dalt, dispositiu i imatge d'MRI. A baix, dispositiu i imatge de CT.

també la direcció d'exploració. Per resoldre tots aquests problemes s'han proposat diferents tècniques de registre.

El *registre* consisteix essencialment a alinear espacialment els conjunts de dades, és a dir, que hi hagi una correspondència entre les dades obtingudes pels diferents dispositius de captació. El model resultat d'aquest procés és el que es coneix com a *model fusionat* o *registrat*. El model registrat es pot interpretar com un model de vòxels en el qual cada vòxel manté *n* valors de propietat que poden provenir de diferents dispositius de captació. Aquest concepte està contraposat al de *model simple*, en el qual cada vòxel té només 1 valor de propietat.

Una vegada s'ha realitzat el registre cal aplicar alguna tècnica de visualització que permeti obtenir la imatge en la qual es representi la informació d'aquest model. Tot i que s'han proposat moltes tècniques per poder visualitzar models de vòxels en els quals només s'ha representat un sol valor de propietat, la visualització de models registrats resulta bastant complicada.

Una de les estratègies que s'ha proposat per resoldre aquest problema es la dels "Miralls Màgics". La idea es col·locar plans al voltant del volum per poder obtenir diferents projeccions del model. Aquesta tècnica és la que implementarem per resoldre el problema de la visualització de models fusionats.

Selecció del punt de vista

Tot i els avantatges que proporciona la tècnica dels miralls màgics, aquesta planteja un nou problema: **on col·locar els plans de projecció**. De fet aquest problema es planteja també en el cas dels models simples, és a dir, **com podem saber quina és la millor vista que es pot obtenir d'un model de volum?**

El punt de vista òptim és aquell que aporta més informació a l'usuari. Generalment l'usuari ha de trobar un punt de vista quasi òptim provant de veure el model des de diferents punts de vista, i escollint el que li sembli que és millor. Això pot ser difícil o lent en alguns casos, i per tant seria interessant poder automatitzar almenys parcialment aquest procés. La importància de la vista dependrà dels interessos de l'usuari.

Assignació de colors

Un dels punts clau en el procés de creació de la imatge final que s'obté del model de volum és l'**assignació de colors als valors de propietat** representats en el model de volum. Donat un punt de vista, l'assignació de colors i transparències del model influeix molt en la qualitat de la visualització, i tampoc **no és fàcil trobar l'assignació més adequada**.

En aquest projecte implementarem un algorisme que permetrà determinar el punt de vista ideal per visualitzar un model de vòxels així com també els punts de vista ideals per als miralls per tal d'aconseguir el màxim d'informació possible del model de vòxels. Aquest algorisme es basa en la teoria de la informació per saber quina és la millor visualització. L'algorisme també permetrà determinar l'assignació de colors òptima per al model de vòxels.

Per fer més entenedora tota la problemàtica que plantegem començarem donant una descripció del procés de visualització d'un model simple. A continuació presentarem una tècnica per visualitzar models fusionats anomenada Miralls Màgics. Finalment, explicarem un mètode per trobar una aproximació al millor punt de vista basat en la teoria de la informació.

2.2 El procés de visualització

En aquest apartat donarem una breu descripció de les diferents etapes que formen el procés de visualització i donarem una classificació dels diferents algorismes de visualització de volums que existeixen.

El procés de visualització permet passar de les dades mesurades a una imatge en pantalla. El procés de visualització segons el model de Haber-McNabb està format per les etapes que es poden veure a la Figura 2.2. Tot seguit expliquem una mica cadascuna de les etapes.

2.2.1 Adquisició de dades

La primera etapa consisteix a l'obtenció de les dades que es volen visualitzar. Aquestes dades es poden obtenir de simulacions o bé de dispositius de captació.

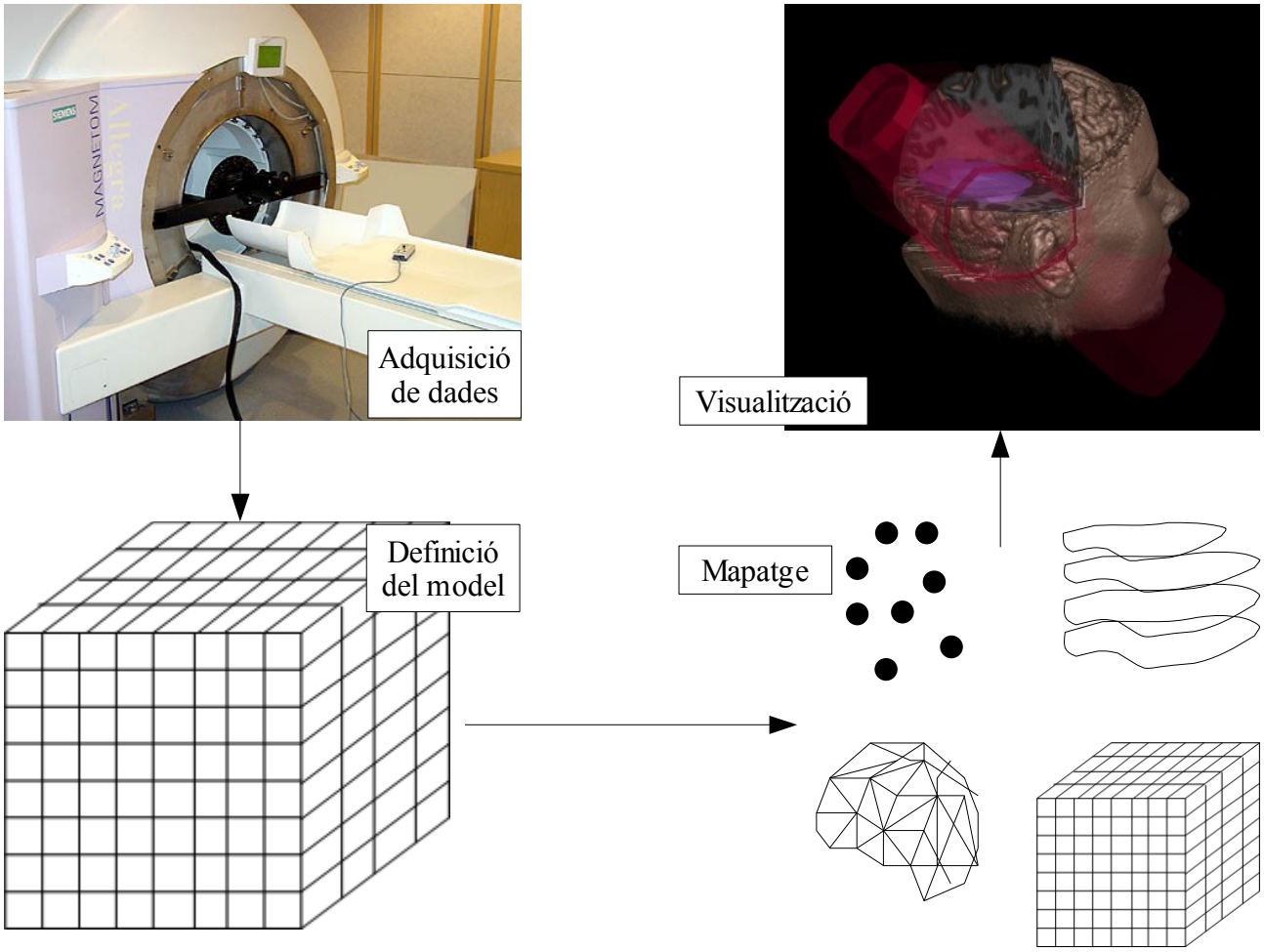


Figura 2.2: Procés de visualització.

Quan es tracta d'aplicacions mèdiques, com en el nostre cas, les dades les proporciona un dispositiu de captació, el qual és capaç de mesurar un o més paràmetres determinats en diferents punts del cos del pacient. Els punts observats sempre segueixen una distribució espacial regular, distribuïts sobre plans paral·lels a algun dels plans de projecció (frontal, coronal o axial). Dins de cada pla els punts formen una graella regular. Com ja hem dit anteriorment, cada dispositiu de captació té les seves característiques particulars. Així, no tenen tots la mateixa resolució a cada pla ni el mateix nombre de plans, i aquests no tenen perquè estar alineats en la mateixa direcció. A més, la zona del cos mesurada també pot ser diferent.

El resultat de la mesura en cada punt és un valor numèric corresponent a la intensitat observada. D'aquest número en diem **valor de propietat**. Les dades es guarden en un fitxer binari en format DICOM. A aquest fitxer binari l'acompanya un fitxer de capçalera de text planer, que ens diu com està estructurat.

2.2.2 Definició del model

Aquesta etapa consisteix a definir el model de representació que es farà servir per a representar les dades que s'han obtingut a l'etapa anterior. En aquest cas ens interessa definir un **model volumètric**.

L'objectiu d'aquest pas és crear un model de dades que faciliti el tractament d'aquestes per part de l'ordinador. En aquest cas hi ha un model que podem considerar un estàndard de facto per al tractament de dades mèdiques: es tracta del **model de vòxels** proposat per Kaufman. Un model de vòxels és una malla regular 3D formada per un conjunt de **cel·les** o **vòxels** on es representen els valors de propietat. Cada cel·la o vòxel és un petit cub o paral·lelepípede, i són tots de la mateixa mida.

Per construir el model de vòxels es parteix del fitxer DICOM proporcionat pel dispositiu de captació. En aquest fitxer es guarden ordenadament les dades distribuïdes en plans paral·lels. Llavors només cal anar recuperant aquests plans i apilar-los amb la separació adequada per aconseguir un quadrícula tridimensional que és el model de vòxels. La informació sobre la separació entre els plans es troba al fitxer de capçalera.

2.2.3 Mapatge

A partir del model de vòxels cal seleccionar la informació que es vol usar per obtenir la imatge final. Aquesta informació anirà associada amb un tipus de primitiva concret.

Els algorismes de visualització seran diferents segons la informació que es deixi passar a l'etapa següent. Si es converteix el model en un conjunt de primitives geomètriques s'obté un model simplificat o reconstruït. Si es manté el model de vòxels original aquesta etapa no fa res i es farà una visualització directa del volum.

Si es vol un model simplificat hi ha tres reconstruccions possibles:

- **Reconstrucció 0D.** En aquest cas s'obtenen punts. Consisteix a agafar tots els valors de propietat del model de vòxels o només alguns.

- **Reconstrucció 1D.** En aquest cas s'obtenen contorns (línies). Consisteix a extreure el contorn a cada llesca. El contorn pot ser el més extern, per exemple la pell, o algun contorn intern, per exemple la superfície de l'os.

- **Reconstrucció 2D.** En aquest cas s'obtenen superfícies. Consisteix a extreure una superfície del model de vòxels. Com abans, pot ser la superfície externa o una superfície interna. Aquesta tècnica se sol basar en **isosuperfícies**.

En tots els casos de reconstrucció es perd informació a canvi de guanyar velocitat en la visualització.

2.2.4 Visualització

Aquesta és l'última etapa abans d'obtenir la imatge final. Consisteix a assignar atributs gràfics a les primitives seleccionades –en el cas de models reconstruïts– o als vòxels –en el cas de mantenir el model de vòxels– i aplicar alguna tècnica que permeti obtenir la imatge final. La tècnica de visualització que s'apliqui dependrà del procés de mapatge aplicat. No és el mateix visualitzar punts que visualitzar contorns, superfícies o vòxels.

Comentarem algunes d'aquestes tècniques més endavant.

El resultat final d'aquesta etapa és la **imatge final**. Una part molt important del treball que hem de desenvolupar per assolir els nostres objectius va lligada amb la visualització; per aquesta raó presentem una classificació dels diferents algorismes que s'han proposat.

Classificació dels algorismes de visualització

En funció de la tècnica de mapatge que s'apliqui durant el procés de visualització els algorismes de visualització es classifiquen en dues grans famílies:

- **Algorismes de visualització de models reconstruïts.**

  En els cas de la visualització de models reconstruïts es poden considerar tres grups diferents:

  - Algorismes que visualitzen punts.
  - Algorismes que visualitzen contorns.
  - Algorismes que visualitzen superfícies.

  De tots aquests algorismes els més populars són els que visualitzen superfícies i dins d'aquest grup el més conegut és l'algorisme de *Marching Cubes*.

- **Algorismes de visualització directa de volums.**

  Aquests algorismes generen una imatge tenint en compte tota la informació que hi ha representada en el model de vòxels. Dins d'aquesta família es distingeixen dos grups diferents:

  - Algorismes d'ordre imatge o tipus *ray casting*.

    Per generar la imatge es llença un raig per cada píxel de la pantalla. Aquest raig s'interseca amb diversos vòxels del volum. Segons com sigui la incidència del raig i les

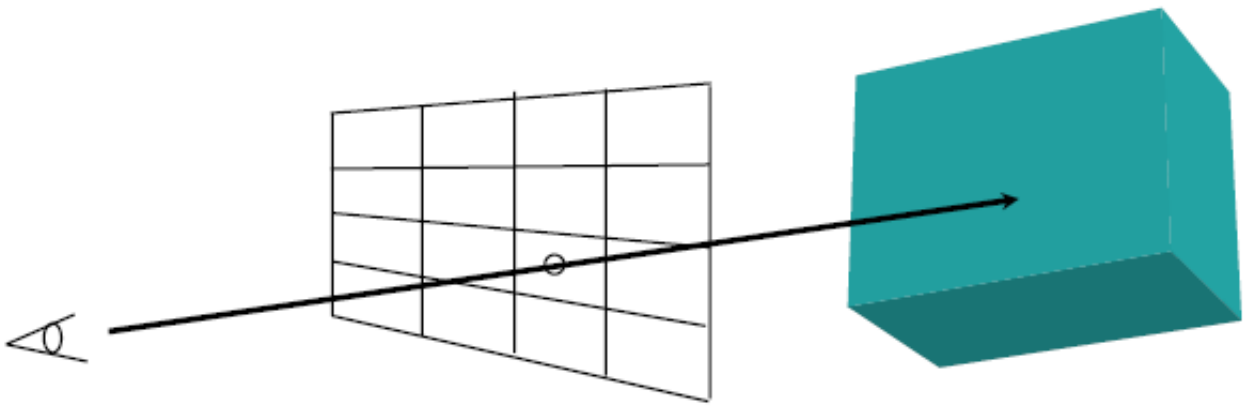


Figura 2.3: Tècnica del ray casting*: per cada píxel de la pantalla es llança un raig que s'interseca amb el model de vòxels.*

propietats gràfiques assignades als vòxels, es calcula el color que cal assignar al píxel de la imatge final. A la Figura 2.3 hi ha una esquema d'aquesta tècnica de visualització.

- Algorismes d'ordre objecte o tipus *splatting*.

  Per generar la imatge final es projecten directament els vòxels sobre la pantalla. Es fa un recorregut del model de vòxels seguint una determinada direcció. El procés es similar al de llençar boles de neu sobre una paret.

En aquest projecte ens hem centrat en els algorismes de visualització directa de volums que segueixen la tècnica de *ray casting*. Hem descartat els algorismes de visualització de models reconstruïts perquè tractaven amb models simplificats del model de vòxels inicial i per tant resultaven una mala opció per visualitzar models fusionats. Una vegada decidits a aplicar algorismes de visualització directa de volums hem escollit els algorismes de tipus *ray casting* ja que són més ràpids i obtenen més qualitat d'imatge que els basats en tècniques d'*splatting*.

2.3 Visualització directa de volums

L'objectiu és obtenir a la pantalla de l'ordinador una imatge que representi tota la informació que hi ha en el model de vòxels. Se sap que el model de vòxels manté un conjunt de valors de propietat que s'ha obtingut d'un pacient a través d'un dispositiu de captació. D'aquests valors de propietat només se sap que estan definits dins d'un rang de valors fixat pel dispositiu de captació i se sap que segueixen una distribució regular.

D'altra banda se sap que la imatge final estarà formada per un conjunt de píxels i que cada píxel porta associat un determinat color, que és el color que es veurà a la pantalla. Així doncs, el problema és **com passar dels valors de propietat representats al model de vòxels als colors dels píxels a la pantalla?**

Per resoldre aquest problema el que es fa és considerar el volum com si fos un gel format per una sèrie de partícules minúscules. Aquestes partícules corresponen als valors de propietat representats al model de vòxels. Es considera que cadascuna d'aquestes partícules és capaç d'emetre llum i també d'absorbir-ne. Les quantitats de llum que poden emetre i absorbir estaran en funció de les propietats que tinguin assignades les partícules. Per determinar quines són aquestes propietats es defineix una funció de transferència.

2.3.1 La funció de transferència

La funció de transferència determina el color i l'opacitat (o grau de transparència) que li correspon a un determinat valor de propietat. Per a fer-ho més entenedor, la funció de transferència determina el color del qual volem pintar una determinada propietat. Aquest color el determina l'usuari. Nosaltres podem dir per exemple que volem veure l'os d'un color blanquinós i totalment opac, o bé que el volem veure blanquinós però semitransparent per tal de poder determinar les seves estructures internes. Podem dir que la pell la volem d'un color rosat, les venes vermelles, etc.

Per definir la funció de transferència cal assignar un color i una opacitat a cada valor de propietat. El color i l'opacitat es representen com un valor RGBA on R representa la component vermella del color

final, G la component verda, B la component blava i A el grau d'opacitat. La component A està definida entre 0 i 1, on 0 és totalment transparent i 1 és totalment opac.

El valor RGBA assignat a una propietat representa la quantitat de llum que es capaç d'absorbir i emetre aquella propietat. Quan un raig travessa els vòxels es pot fer una interpolació entre els valors per decidir el color que s'agafa per aquell punt, o es pot agafar simplement el més proper.

El problema a l'hora de definir la funció de transferència és que **és molt difícil triar la funció de transferència adequada**. Això és perquè no sabem quines estructures hi ha en el model de vòxels. No hi ha un estàndard per als valors de propietat corresponents a cada tipus de teixit, sinó que això depèn del dispositiu. Per tant, és impossible definir una funció de transferència estàndard que pugui anar més o menys bé amb qualsevol model de vòxels.

Així doncs, tenim dues solucions possibles per trobar la funció de transferència més adequada per un model de vòxels donat:

1. **Prova i error.** Aquesta solució consisteix a definir una funció de transferència inicial i anar-la modificant fins a trobar-ne una que vagi bé pels interessos de l'usuari. És la solució més fàcil d'implementar.

2. **Trobar automàticament una funció de transferència que s'ajusti al model.** Aquesta solució consisteix a que sigui l'ordinador el que defineixi la funció de transferència inicial i llavors l'usuari pugui modificar-la si vol. Aquesta funció de transferència inicial s'ha d'ajustar bé al model. Per tal de fer això cal poder detectar quines estructures hi ha al model i saber els seus rangs de valors corresponents. Això evidentment no és fàcil, però es poden aplicar algunes tècniques basades en la teoria de la informació per segmentar les imatges obtenint-ne les estructures [4]. Un cop trobats els rangs de valors de les estructures només cal assignar un valor RGBA diferent a cada rang.

En aquest projecte implementarem la primera solució i proposarem una implementació de la segona.

Un cop definida la funció de transferència per determinar els colors de la imatge final que obtindrem a partir del nostre algorisme de visualització de volums, ens cal definir la tècnica de composició de colors que aplicarem.

2.3.2 Composició de colors

Un cop definida la funció de transferència i per tant assignats els valors RGBA als diferents valors de propietat representats en el model de vòxels, per generar la imatge final el que hem de fer és llançar un raig de llum per cada píxel de la pantalla. Aquest raig és una línia recta que travessarà algunes de les partícules que composen el volum. El problema que ara es planteja és com determinar el color final que s'obté tenint en compte la quantitat de llum que absorbeix i emet cadascuna de les partícules intersecades, és a dir, tenint en compte el valor RGBA que té assignada cadascuna de les propietats.

Per resoldre aquest problema s'aplica una equació simplificada de la teoria de transport de la llum. La base d'aquesta simplificació és no considerar el raig de llum com una línia continua sinó considerar-lo com una línia discreta.

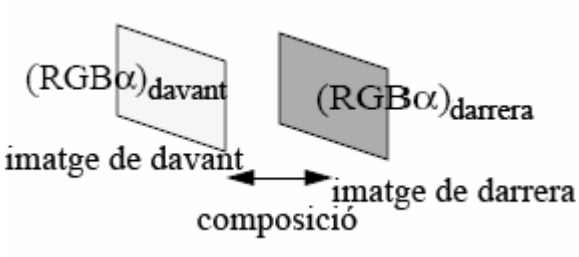
- Els colors i opacitats dels píxels del darrera són atenuats per les opacitats dels píxels del davant:

$$rgb = rgb_{darrera} \cdot \alpha(1-\alpha_{davant}) + rgb_{davant} \cdot \alpha_{davant}$$
$$\alpha = \alpha_{darrera}(1-\alpha_{davant}) + \alpha_{davant}$$

- Fent servir:

$$rgb_{darrera} = rgb_{darrera} \cdot \alpha_{darrera}$$
$$rgb_{davant} = rgb_{davant} \cdot \alpha_{davant}$$

obtenim 2 equacions recursives que es poden fer servir per compondre qualsevol nombre d'objectes de davant cap endarrere:



$$rgb_{davant} = rgb_{darrera}(1-\alpha_{davant}) + rgb_{davant}$$
$$\alpha_{davant} = \alpha_{darrera}(1-\alpha_{davant}) + \alpha_{davant}$$

- La visualització directa de volums utilitza aquesta expressió recursiva per combinar (compondre) les mostres preses al llarg del raig

Aplicant aquesta equació de forma recursiva obtenim la imatge final.

Aquesta equació és la base de tots els algorismes de visualització directa de volums. La diferència que hi ha entre els diferents algorismes rau en la forma de prendre les mostres sobre el raig de llum que es llança, en l'ordre en què es prenen les mostres o en l'ordre en què es fa l'assignació de colors i opacitats.

2.4 Ray casting

Aquí explicarem la tècnica de *ray casting* de forma general, tant per visualitzar polígons com per fer visualització directa de volums. L'explicació general i l'específica del *ray casting* geomètric està extreta de [5], on hi ha dels algorismes de *ray tracing* i *radiositat*.

Podem definir un raig com una línia recta amb un punt de partida fix a partir del qual s'estén en una direcció donada cap a l'infinit. Quan encenem una llum, els raigs es posen en moviment rebotant en tots els objectes. Si un raig toca un objecte poden passar vàries coses: es pot reflectir, es pot absorbir o es pot transmetre. Fins i tot hi ha casos en què es pot absorbir i després transmetre (objectes fluorescents).

Si tenim en compte que els raigs es poden representar com a rectes i que els objectes de l'escena es poden modelar matemàticament, llavors afegint els coneixements de la física que ens diuen com es calculen les reflexions i les refraccions d'un raig podríem modelar el comportament de la llum únicament coneixent la posició de totes les fonts.

El que es pretén fer és seguir el camí dels raigs procedents de les fonts de llum i veure la seva influència en cadascun dels objectes de l'escena. Com que és molt difícil (i costós) seguir el camí de cada raig (dels quals d'altra banda n'hi ha infinits) a partir de la font de llum, es realitza el procés invers: es selecciona un centre de projecció (el punt de vista) i una finestra en un pla arbitrari (el que correspondria a la pantalla), es divideix la finestra en píxels depenent de la resolució que volem tenir i després per a cadascun d'aquests píxels es traça (o llança) un raig amb origen al centre de projecció, cap a l'escena (raig de visió). Quan el raig col·lideix amb un objecte llavors es calcula el color del píxel corresponent. Llavors es tracen altres raigs, de manera recursiva, amb l'origen al punt d'intersecció i en la direcció de cada una de les fonts de llum. D'aquesta manera es calcula de quina manera la font contribueix a la il·luminació del píxel.

D'aquesta manera, s'aconsegueix considerar únicament aquells raigs que arriben directament als nostres ulls i que contribueixen a la il·luminació de l'escena.

Això és per a un *ray casting* geomètric (els objectes de l'escena són polígons). En el cas del *ray casting* volumètric no hi ha fonts de llum i el raig travessa una sèrie de vòxels i calcula el color del píxel fent una composició dels valors RGBA que retorna la funció de transferència per cada valor de propietat. No existeixen "col·lisions".

2.4.1 Elements d'un traçador de raigs

Fonamentalment, per modelar el comportament de la llum en aquest cas, necessitem tres elements: l'observador, els objectes de l'escena i les fonts d'il·luminació. Ara veurem breument cadascun d'aquests elements.

L'observador

En el nostre cas, l'observador es pot considerar com una càmera amb la qual es pren la foto de l'escena. A partir d'un punt d'observació, i a una distància determinada, es col·loca un pla quadriculat (la pantalla) perpendicular a l'escena que volem representar. Per cada un dels quadradets del pla (píxels) es veurà un tros d'escena, que serà el que representarem al píxel corresponent de la pantalla.

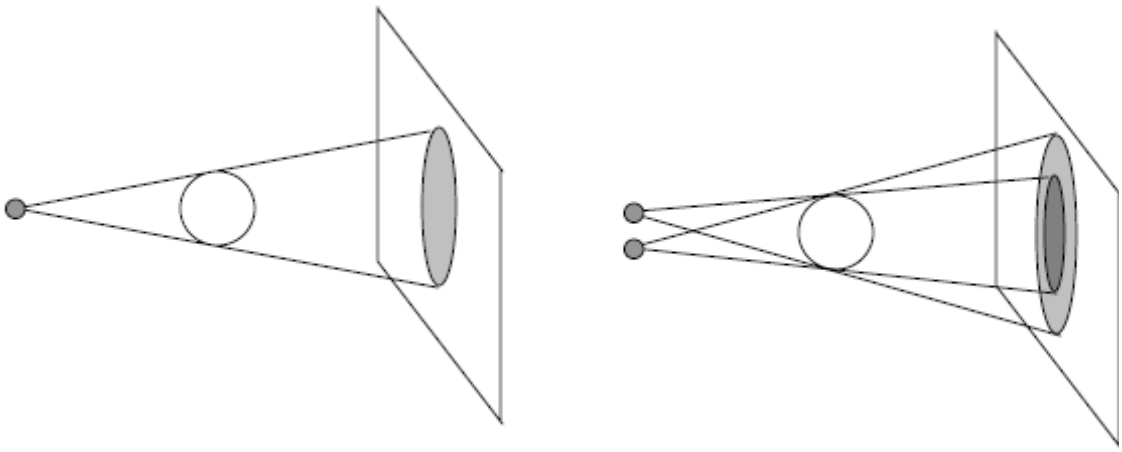


Figura 2.4: Esquerra: amb una font puntual s'aconsegueix només ombra. Dreta: Amb una font múltiple s'aconsegueix ombra i penombra.

Aquest procés no és nou, ja que és exactament el que s'utilitzava durant el Renaixement per la representació en perspectiva. Per exemple, l'artista alemany Albrecht Dürer utilitzava una quadrícula transparent (de fet era un cristall) i mirava a través d'ella per dibuixar les seves pintures. La tela del quadre estava també dividida en el mateix número de quadrícules i l'artista mirava per un petit forat i així reduïa el punt de vista a una única posició. Es pot veure un dibuix d'això a la Figura 2.5.

Els objectes

Per aquest tipus d'algorismes, els objectes més adequats són aquells que es poden representar fàcilment amb expressions matemàtiques (això és fonamentalment la causa que la majoria d'imatges d'exemple del *ray casting* representin esferes, cilindres o cubs). Encara que de fet es pot utilitzar qualsevol representació amb la qual es pugui calcular (més o menys ràpidament) la seva intersecció amb un raig (per exemple, un model de vòxels).

Figura 2.5: Tècnica de dibuix d'Albrecht Dürer.

Les fonts de llum

El tipus de fonts de llum és molt variat (llum solar, bombetes, fluorescents, foc, etc.). D'elles ens interessen les seves principals propietats: el color, la posició, la intensitat, la grandària i la forma.

Les dues simplificacions més habituals al *ray casting* són:

1. Suposar que la llum que s'emet és d'un únic color (en realitat es composa d'una combinació d'un ampli interval).

2. Suposar que la llum prové d'un punt infinitament petit.

La segona simplificació fa que els objectes no mostrin penombres, sinó que únicament mostraran ombres ben definides (la qual cosa no és gaire habitual a la realitat). Si es vol modelar l'efecte de la penombra es pot modelar la font de llum com un conjunt (o matriu) de fonts puntuals. Podem veure aquest efecte a la Figura 2.4.

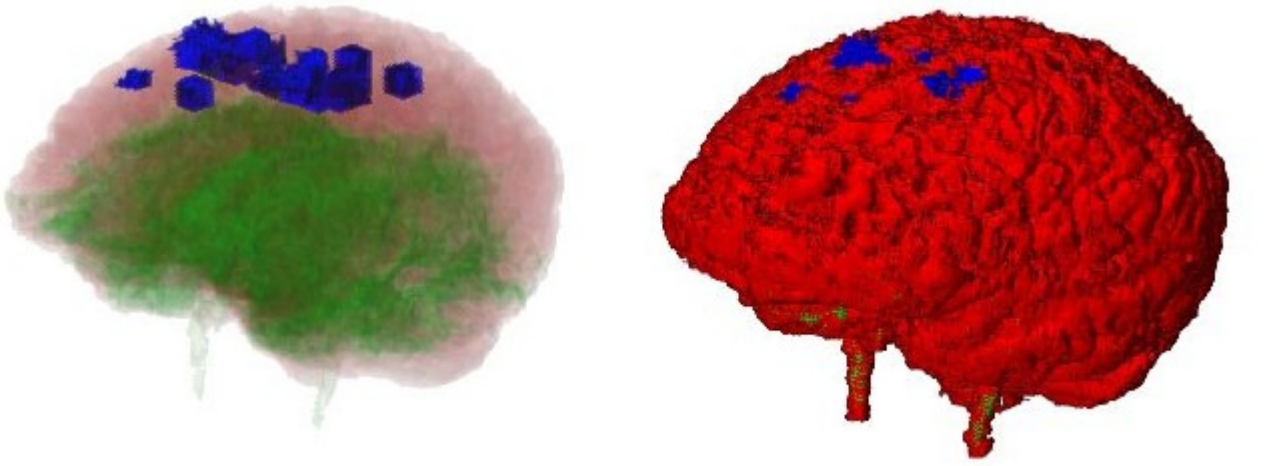


Figura 2.6: Esquerra: visualització del volum amb una opacitat molt baixa; és difícil apreciar els detalls i fer-se una idea de l'estructura espacial de les dades. Dreta: visualització del volum amb una opacitat molt alta; no es veu l'interior del volum.

Cal recordar que al *ray casting* volumètric, que és el que ens interessa, no es fan servir fonts de llum, sinó que s'agafen directament els colors dels vòxels (donats per la funció de transferència) i es considera que hi ha la mateixa il·luminació a tots els punts de l'espai.

2.4.2 Tipus de raigs

El raigs amb què es treballa poden ser dels següents tipus:

- **Raig de visió:** És el que s'envia des del punt de visió fins a l'escena passant per cadascun dels píxels del rectangle de la finestra. Serveix per calcular el color del píxel corresponent a la imatge final. En el cas del *ray casting* volumètric travessa una sèrie de vòxels i compon els seus valors RGBA corresponents per calcular el color del píxel.

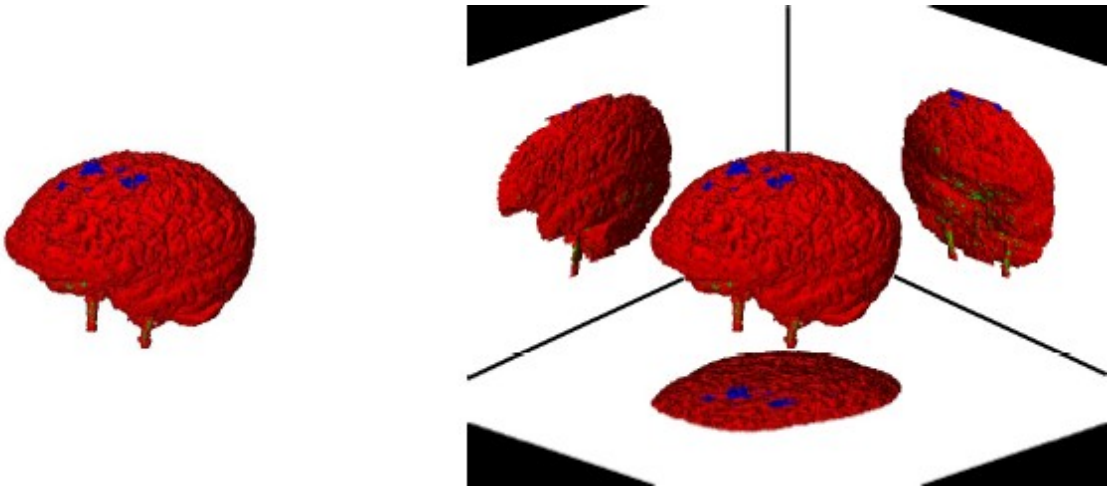


Figura 2.7: Esquerra: visualització directa d'un volum sense vistes múltiples. Dreta: visualització directa d'un volum amb vistes múltiples.

- **Raig d'il·luminació (només al *ray casting* geomètric):** Una cop el raig de visió s'interseca amb un objecte de l'escena, s'envia un raig d'ombra des d'aquest punt d'intersecció a cada una de les fonts de llum definides a l'escena. Si el raig arriba a la font directament (sense travessar el propi objecte), llavors es considera la influència d'aquesta font per calcular la il·luminació final del punt.

- **Raig reflectit (només al *ray casting* geomètric):** Amb la tècnica de *ray casting* només hi ha reflexió difusa (i només en el cas del geomètric), que es produeix en totes direccions.

2.5 Limitacions de la visualització directa de volums. Solució basada en múltiples vistes

La visualització directa de volums també té algunes mancances, ja que si els nivells d'opacitat que ens retorna la funció de transferència són molt petits pot ser difícil apreciar els detalls de les zones d'interès. Contràriament, si l'opacitat és molt alta no podrem veure gaire més enllà de la superfície del volum, i l'interior quedarà ocult (vegeu la Figura 2.6 per veure els 2 casos). Per solucionar aquests problemes afegirem informació addicional a la visualització que ajudarà a saber on estan situades les dades d'interès. Aquesta informació addicional serà en la forma de vistes alternatives del volum.

Amb una sola imatge quieta és difícil apreciar correctament la profunditat, ja que la veiem en un dispositiu 2D, una pantalla. Per aquest motiu K. Rehm i altres autors han proposat projectar les regions d'interès a les parets d'un entorn en forma de cub, per ajudar a veure el context i la situació d'aquestes regions. Amb aquestes vistes alternatives és molt més fàcil deduir la disposició espacial de les dades (vegeu la Figura 2.7 per a la comparació). Amb aquesta idea farem una extensió de la visualització directa de volums anomenada "Miralls Màgics", que explicarem més endavant.

A la resta d'aquest apartat i el següent, referent als Miralls Màgics, la informació està treta principalment de [6].

També es pot trobar un estudi que explica perquè es fan servir plans al voltant del volum per les vistes addicionals i no un altre sistema –com per exemple més finestres– a [7].

Continuant amb les múltiples vistes, les imatges dels plans són projeccions ortogonals del volum tal com es veuria des de darrera de cada pla (vegeu la Figura 2.7, dreta). Aquestes es poden generar amb la mateixa tècnica de *ray casting* i després enganxar-les com a textures als plans. Per aquesta aplicació la projecció ortogonal s'ha demostrat que és més útil que la perspectiva, ja que la distorsió de la perspectiva no ajuda l'usuari a interpretar la localització de les dades d'interès.

Aquests plans tenen característiques diferents a les que pot tenir un mirall, per exemple. El que es veu en un mirall és diferent segons el punt de vista de l'usuari; en canvi, en aquests plans es veu sempre la mateixa informació independentment del punt de vista de l'usuari, que simplement veurà la informació de manera diferent. Els plans tenen aquestes propietats perquè d'aquesta manera resulten més útils.

Els beneficis que aporten les vistes múltiples són els següents:

- La localització de les regions d'interès és òbvia fins i tot en imatges quietes, ja que el context tridimensional es pot reconèixer més fàcilment quan tenim múltiples vistes a la mateixa

escena. Això es nota especialment quan els valors de transparència són molt alts (vegeu la Figura 2.8, esquerra i compareu-la amb la Figura 2.6, esquerra).

- El problema de l'oclusió es resol bastant satisfactòriament. A la Figura 2.8, dreta, per exemple, s'ha tallat un bloc de dades del darrere, però gràcies a les vistes múltiples les localitzacions de les dades d'interès encara són visibles.

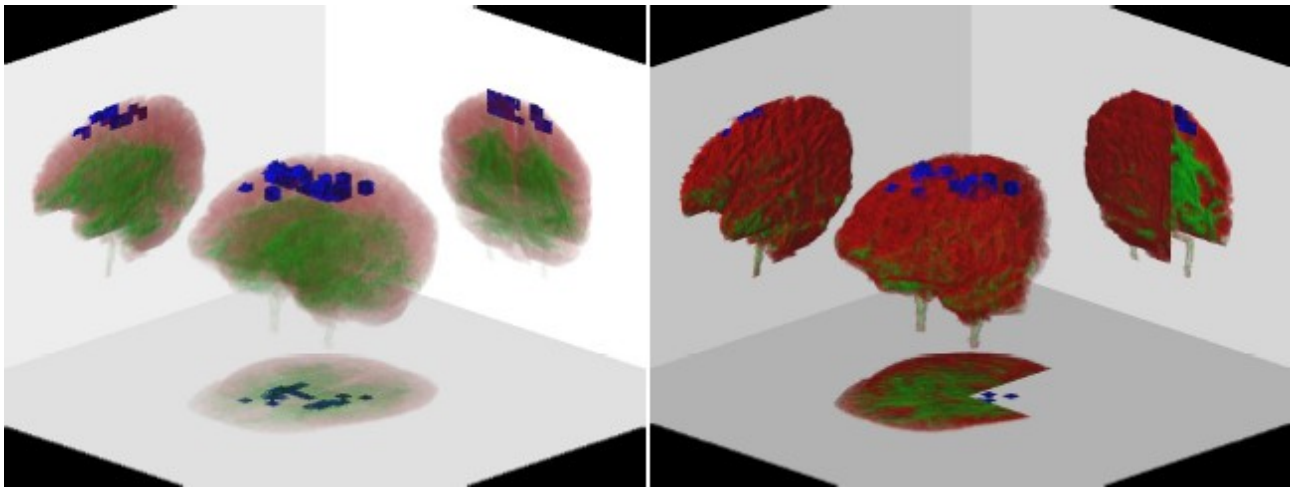


Figura 2.8: Esquerra: utilitzant múltiples vistes, es pot deduir la localització exacta de les activacions fins i tot amb nivells alts de transparència. Dreta: fins i tot les regions ocultes com el tall del darrera són òbvies amb l'ajuda de vistes múltiples.

2.6 Miralls Màgics

Les vistes múltiples aporten bons beneficis, però encara es poden millorar. Tenint en compte que els plans al voltant del volum no tenen unes característiques físiques reals (no són miralls), podem fer servir tècniques de visualització diferents per generar les textures dels plans que per dibuixar el volum. La versió estesa de les vistes múltiples s'anomena Miralls Màgics i aquestes són les seves propietats:

- *"Projecció"* vs. *"Reflexió"*: Les textures per les múltiples vistes són creades amb la tècnica del *ray casting* fent servir un punt de vista des de darrera dels plans. Amb això obtenim un efecte semblant a la reflexió. Però podem posar fàcilment el punt de vista a l'altra banda del volum i d'aquesta manera obtenir les projeccions contràries (des de davant, des de dalt i des de la dreta).

- Es poden fer servir funcions de transferència diferents per cadascun dels Miralls Màgics (vegeu la Figura 2.10). En un mirall podem destacar unes certes propietats i en un altre mirall unes altres propietats.

- Es poden afegir imatges obtingudes amb tècniques de visualització diferents de la visualització directa de volums per proporcionar informació complementària. Per exemple, en un mirall podem posar una imatge del contorn del volum juntament amb les dades d'interès ressaltades, com a la Figura 2.9.

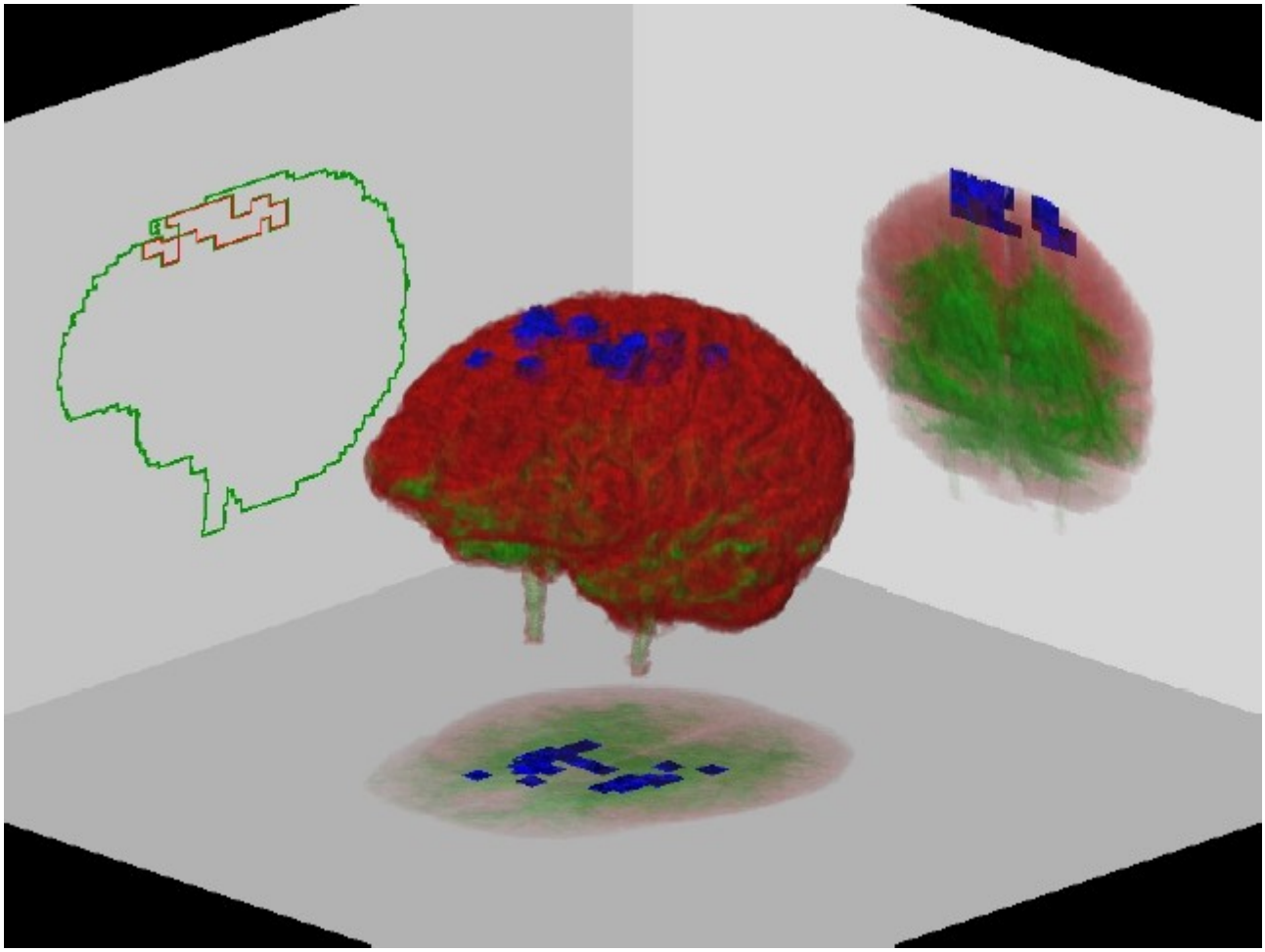


Figura 2.9: Una altra possibilitat dels Miralls Màgics: al mirall de l'esquerra es visualitza el contorn del volum i de les propietats destacades.

Totes aquestes millores es poden combinar entre elles per oferir més versatilitat als Miralls Màgics.

2.7 Miralls Màgics per a models fusionats

A l'hora de realitzar visualitzacions de models fusionats ens trobem amb els mateixos problemes que a l'hora de visualitzar els volums simples. Hem de ser capaços de definir una funció de transferència que ens indiqui quines propietats visuals té una propietat determinada. Cal tenir en compte que en tractar-se de volums fusionats no tindrem només una sola propietat sinó que en tindrem més d'una.

Si considerem que el volum registrat representa la informació de dos volums v_1 i v_2, pels quals hem definit les funcions de transferència f_1 i f_2, les principals opcions de pintat que tenim són les que es presenten a continuació:

1. Pintar el volum fusionat usant la funció de transferència f_1. En aquests cas la imatge que es genera és la mateixa que obtindrem si haguéssim pintat directament el volum v_1.

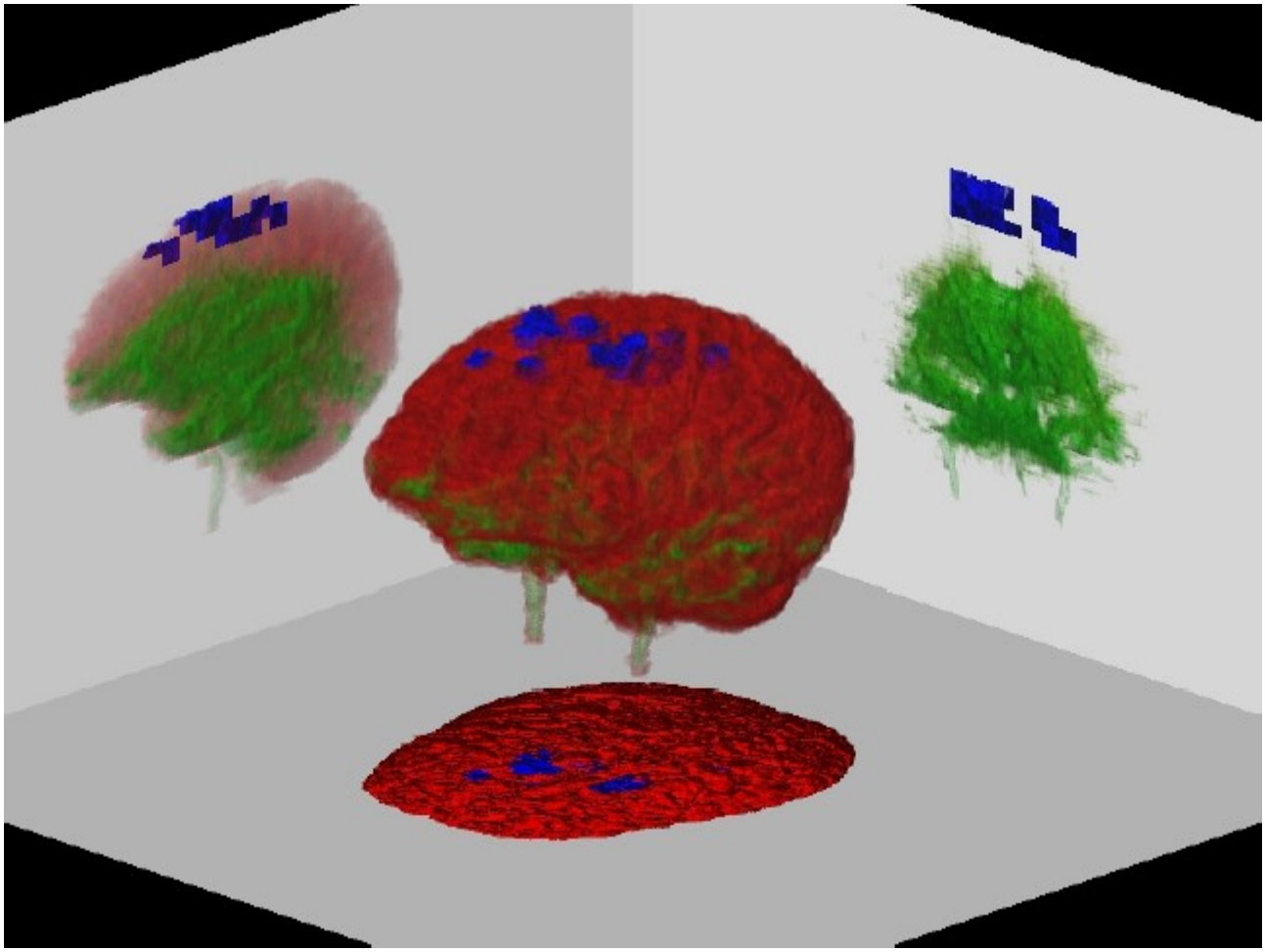


Figura 2.10: Miralls Màgics: es poden fer servir funcions de transferència diferents a cada lloc (el volum central i els miralls). En aquesta imatge hi ha 4 funcions de transferència diferents, on el que varia és el grau d'opacitat de la zona vermella.

2. Pintar el volum fusionat usant la funció de transferència f_2. En aquests cas la imatge que es genera és la mateixa que obtindrem si haguéssim pintat directament el volum v_2.

Aquestes dues primeres opcions no tenen massa sentit, ja que només estem considerant informació d'un únic volum.

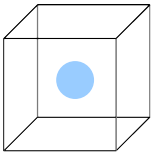
3. La tercera opció es calcular un nou valor de propietat a partir de les mostres representades en el vòxel (mitjana, màxim, mínim, etc.). Pel nou valor calculat es defineix una nova funció de transferència. El volum fusionat es pinta usant aquesta nova funció.

4. La darrera possibilitat és aplicar una combinació de valors d'un model i de l'altre.

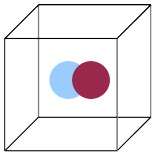
Cap d'aquestes opcions no sembla gaire bona, però aprofitant les característiques dels Miralls Màgics es pot trobar una solució al problema de visualitzar models registrats: **amb els Miralls Màgics podem visualitzar propietats diferents a cada lloc**.

Per exemple, si tenim un model fusionat amb tres propietats podríem veure les tres propietats juntes al volum central (usant la 3a o la 4a opció de l'apartat anterior), una propietat al mirall de l'esquerra, una altra al mirall de la dreta i l'altra al de baix.

També podríem veure totes les propietats a tots els miralls, però amb diferents combinacions de les funcions de transferència.



Vòxel d'un model simple: només hi ha una propietat



Vòxel d'un model registrat: hi ha dues o més propietats

Figura 2.11: Tipus de vòxels.

2.8 Implementació dels Miralls Màgics: on col·locar els miralls

Quan s'aplica la tècnica dels Miralls Màgics cal decidir on posar la càmera principal i també on posar els miralls. Depenent d'aquests posicionaments l'usuari aconseguirà una vista més o menys útil.

El concepte d'*utilitat* és subjectiu i variable segons el cas, però en general podem considerar que la vista més útil és aquella que aporta **més informació** a l'observador. Basant-nos en aquesta idea, podem aplicar alguns conceptes de la Teoria de la Informació per determinar quanta informació podem obtenir des d'un determinat punt de vista, i per tant tenir una mesura de la seva utilitat.

Així doncs, el que caldria és calcular la informació obtinguda des de cada punt de vista possible i escollir el que n'aporti més per col·locar-hi la càmera principal. Després, en alguns dels altres s'hi posarien els miralls. El problema és que hi ha infinits punts de vista possibles i no els podem provar tots. Per tant, el que caldrà fer és escollir un conjunt finit de punts de vista i aplicar els càlculs a cadascun d'ells. Per aconseguir una bona aproximació al punt de vista òptim els punts de vista considerats han d'estar distribuïts uniformement al voltant del volum. Això equival al problema de distribuir punts uniformement sobre la superfície d'una esfera.

2.8.1 Distribució uniforme de punts sobre la superfície d'una esfera

El problema de distribuir punts uniformement sobre la superfície d'una esfera és un problema resolt i se'n pot trobar més informació a [8].

Només hi ha 5 solucions possibles a aquest problema, que corresponen als 5 políedres regulars que existeixen[1]. Concretament, els punts s'han de col·locar als vèrtexs d'un políedre regular.

1 En un vèrtex poden coincidir 3, 4 o 5 triangles equilàters, 3 quadrats o 3 pentàgons. 3 hexàgons s'aplanarien.

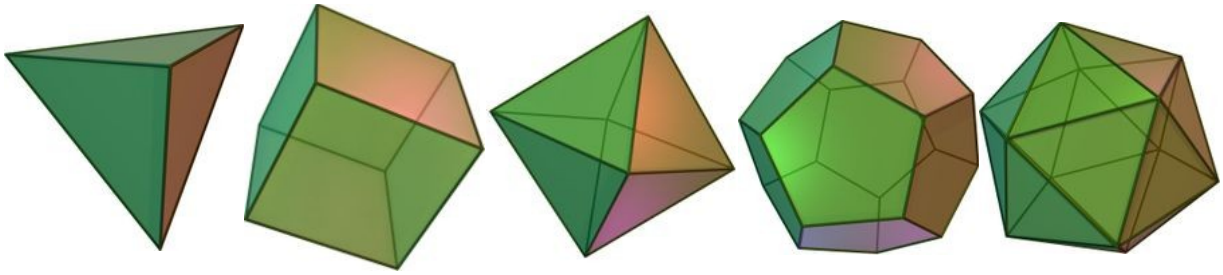


Figura 2.12: Políedres regulars.

Els políedres regulars són el tetràedre, el cub, l'octàedre, el dodecàedre i l'icosàedre (regulars). Per tant, el nombre de punts de vista a considerar serà 4, 6, 8, 12 o 20. Com més punts es calculin més probabilitats hi haurà de trobar l'òptim, però també serà més costós en temps. Per tant, caldrà triar el nombre de punts més adequat segons les necessitats de l'usuari.

Un cop triats els punts de vista possibles cal mesurar la quantitat d'informació que s'obté de cadascun. La manera de fer-ho és mitjançant un concepte bàsic de la Teoria de la Informació: l'entropia.

2.8.2 Mesura d'informació

Sigui $\mathcal{X}$ un conjunt finit, sigui X una variable aleatòria prenent valors x a $\mathcal{X}$ amb una distribució $p(x)=Pr[X=x]$. Així mateix, sigui Y una variable aleatòria prenent valors y a $\mathcal{Y}$. L'*entropia de Shannon H*(*X*) d'una variable aleatòria *X* es defineix com $H(X)=-\sum_{x\in X} p(x)\log p(x)$ i mesura la incertesa mitjana de la variable aleatòria *X*. Si els logaritmes es prenen en base 2, l'entropia s'expressa en bits. L'*entropia condicional* es defineix com $H(X|Y)=-\sum_{x\in X,\, y\in \mathcal{Y}} p(x,y)\log p(x|y)$, on $p(x,y)=Pr[X=x, Y=y]$ és la probabilitat conjunta i $p(x|y)=Pr[X=x|Y=y]$ és la probabilitat condicional. L'entropia condicional $H(X|Y)$ mesura la incertesa mitjana associada amb *X* si sabem el resultat d'*Y*. La *informació mútua* entre *X* i *Y* es defineix com $I(X,Y)=H(X)-H(X|Y)=H(Y)-H(Y|X)$ i mesura la informació compartida entre *X* i *Y*.

Veiem ara les definicions d'*entropy rate* i *excess entropy*. Donada una cadena $\ldots X_{-2}X_{-1}X_0X_1X_2\ldots$ de variables aleatòries X_i prenent valors a $\mathcal{X}$, un bloc de L variables aleatòries consecutives es denota per $X^L=X_1\ldots X_L$. La probabilitat que hi hagi un bloc concret x^L es denota per $p(x^L)$. L'entropia de Shannon de seqüències de longitud L o ***L-block entropy*** es defineix com

$$H(X^L)=-\sum_{x^L\in X^L} p(x^L)\log p(x^L),$$

on la suma passa per tots els blocs possibles de longitud *L*. L'***entropy rate*** es defineix com

$$h^x=\lim_{L\to\infty}\frac{H(X^L)}{L}=\lim_{L\to\infty} h^x(L),$$

on $h^x(L)=H(X_L|X_{L-1}X_{L-2}\ldots X_1)$ és l'entropia d'un símbol condicionada per un bloc de L − 1 símbols adjacents. L'*entropy rate* d'una seqüència mesura la quantitat mitjana d'informació per símbol *x* i la consecució òptima per qualsevol algorisme de compressió possible.

Una mesura complementària de l'*entropy rate* és l'*excess entropy*, que és una mesura de l'*estructura* d'un sistema. L'***excess entropy*** es defineix com

$$E=\sum_{L=1}^{\infty}(h^x(L)-h^x)=\lim_{L\to\infty}[H(X^L)-h^x L]$$

i captura com $h^x(L)$ convergeix cap al seu valor asimptòtic h^x. Així, quan tenim en compte un nombre petit de símbols en el càlcul de l'entropia, el sistema sembla més aleatori del que és realment. Aquesta aleatorietat excessiva ens diu quanta informació addicional cal guanyar sobre les configuracions per tal de revelar la incertesa real h^x.

L'aplicació d'aquests conceptes al projecte consistirà a calcular l'*entropy rate* i l'*excess entropy* per cada punt de vista per aconseguir una mesura de la quantitat d'informació que s'obté amb cadascun.

Per fer els càlculs es col·locarà un pla a cada punt de vista on es representarà el que es veuria des d'una finestra en aquella posició. Des de cada pla es llançaran n^2 raigs distribuïts uniformement i a cada raig es prendran *m* mostres distribuïdes uniformement. Cadascuna d'aquestes mostres serà un dels valors x_i dels blocs de longitud *L* que es faran servir en els càlculs.

Aquests càlculs no els podem fer sobre el model de vòxels original perquè tenen un cost computacional molt elevat. La quantitat de memòria que necessita l'histograma utilitzat per calcular l'entropia és $O(N^L)$, on *N* és el nombre de valors de propietat diferents i *L* la longitud de bloc. Tenint en compte que els models de vòxels tenen habitualment un rang de valors de propietat de 256 valors, es fa evident que el problema és intractable quan comença a créixer la longitud de bloc.

Per tant, caldrà simplificar el model de vòxels perquè tingui un rang de valors de propietat molt més petit. Per fer això caldrà aplicar algun mètode de segmentació que trobi les estructures del model de

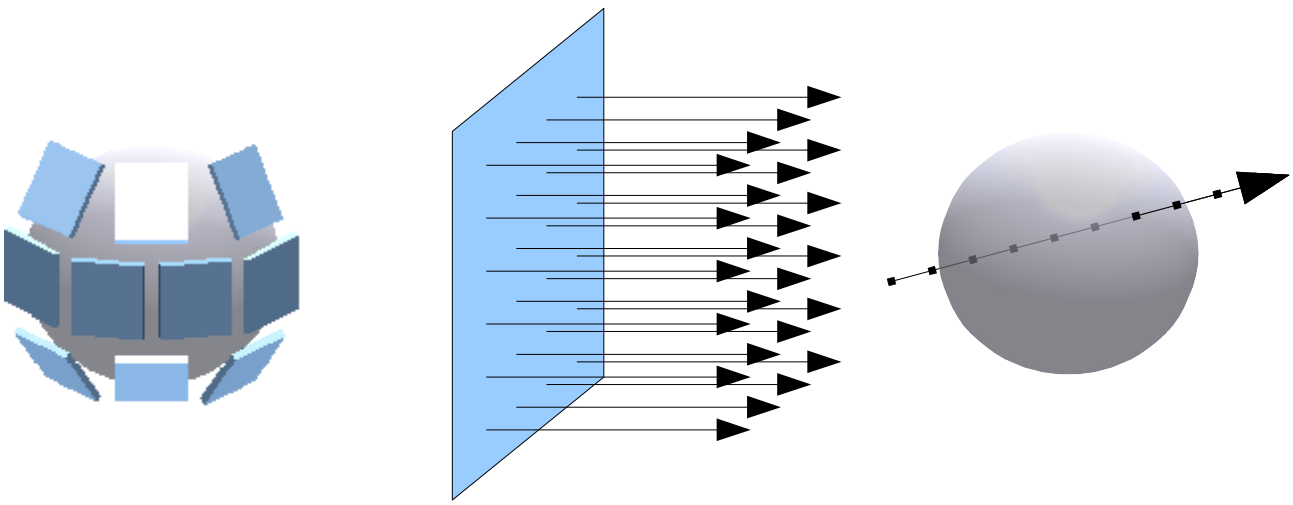


Figura 2.13: Esquema de càlcul de la quantitat d'informació. Esquerra: s'envolta el model amb plans. Centre: es llancen n^2 *raigs des de cada pla. Dreta: a cada raig es prenen* m *mostres.*

vòxels i les faci homogènies, és a dir, que faci que cada estructura tingui un sol valor de propietat. També és important que el rang sigui compacte, és a dir, que els valors siguin seguits, sense "forats", i que comenci a 0, per facilitar els càlculs; això també ho farà el mètode de segmentació. Amb això s'aconseguirà un rang de valors de propietat molt més petit[2] i l'histograma tindrà una mida raonable. El mètode de segmentació es basarà en el mateix principi de l'*excess entropy*.

2.9 Resum

En aquest capítol hem repassat els aspectes teòrics més importants relacionats amb els problemes que volem resoldre en el projecte. Hem presentat els problemes i també les tècniques que proposem per resoldre'ls. Aquests problemes són:

- La visualització de models fusionats, com veure en una mateixa imatge informació corresponent a dos models de volum diferents. Per tractar aquest problema proposem usar la tècnica dels Miralls Màgics.

- La selecció del punt de vista, com determinar quina és la imatge que ens dóna la millor vista del model que volem visualitzar en pantalla. Aquest problema el tractarem per models simples. Per resoldre aquest problema proposem usar eines de teoria de la informació que ens permetin quantificar la informació que podem obtenir des de diferents posicions. Ens centrarem en l'*excess entropy*.

- L'assignació de colors, com determinar de quins colors s'han de pintar les diferents propietats que estan representades en el model de volum que volem visualitzar. Proposarem també l'aplicació de mesures de teoria de la informació.

Volem remarcar que en aquest capítol només s'han tractat alguns dels aspectes per poder entendre la problemàtica que tractaríem. Al llarg dels altres capítols i a mesura que es donin detalls de la implementació si és necessari s'entrarà amb més detall en els aspectes teòrics.

2 El nombre de valors exacte depèn del model, però un nombre raonable pot ser de 6 o menys valors.

3 Fonaments pràctics

En aquest capítol explicarem els fonaments pràctics, és a dir, les eines i biblioteques fetes servir en la implementació del projecte. Per cada una de les eines donarem una breu descripció i explicarem com s'ha usat en el projecte.

3.1 Qt

La informació d'aquest apartat està treta del *Qt 4.1 Whitepaper*, disponible des de la pàgina web de Trolltech, juntament amb la documentació completa de Qt, programes d'aprenentatge i les biblioteques per a diferents plataformes [9].

3.1.1 Descripció

Les Qt són unes biblioteques per C++ que permeten desenvolupar aplicacions amb interfície gràfica d'usuari (GUI, *Graphical User Interface*). Són totalment orientades a objectes i ofereixen un conjunt de classes que faciliten la programació d'aplicacions gràfiques, multiplataforma i multilingües.

Les llibreries Qt proporcionen un conjunt de *widgets* que permeten fer les GUIs. També tenen un sistema de comunicació entre objectes anomenat *"signals i slots"*. També hi ha un model d'esdeveniments convencional que permet gestionar els clics del ratolí, les tecles premudes, etc. Les Qt permeten crear tots els elements de les GUIs d'avui en dia, com menús, menús contextuals i barres d'eines.

Juntament amb les Qt hi ha un programa, el Qt Designer, que permet dissenyar gràficament les interfícies d'usuari i veure com queden amb diferents estils. Un altre programa, el Qt Linguist, permet traduir les aplicacions que fan servir Qt a altres idiomes.

Les Qt suporten gràfics en 2D i en 3D, permeten connexions a bases de dades independents de la plataforma, i poden emular l'aparença de diversos sistemes operatius i entorns gràfics.

3.1.2 Widgets

Les Qt disposen d'una àmplia varietat de *widgets*, que són elements visuals que es combinen per crear interfícies d'usuari. Exemples de *widgets* són botons, menús, barres de desplaçament i finestres, entre d'altres. Tots els *widgets* de Qt es poden fer servir com a controls i com a contenidors, i es poden crear nous *widgets* com a subclasses dels existents. A la Figura 3.1 es poden veure imatges de diferents *widgets* construïts a partir de *widgets* bàsics.

A les Qt hi ha la classe **QWidget**, que és el *widget* més bàsic, i la resta de *widgets* són subclasses d'aquesta.

Un *widget* pot contenir qualsevol nombre de *widgets* fills, que són dibuixats a l'interior del *widget* pare. Els *widgets* fills s'organitzen dins del pare fent servir diferents tipus de *layouts* bàsics que es poden combinar per aconseguir distribucions més complexes. El *widget* de més alt nivell és el que no

té pare, generalment la finestra principal. Quan s'inhabilita, s'amaga o es destrueix un *widget* s'aplica la mateixa acció a tots els fills recursivament.

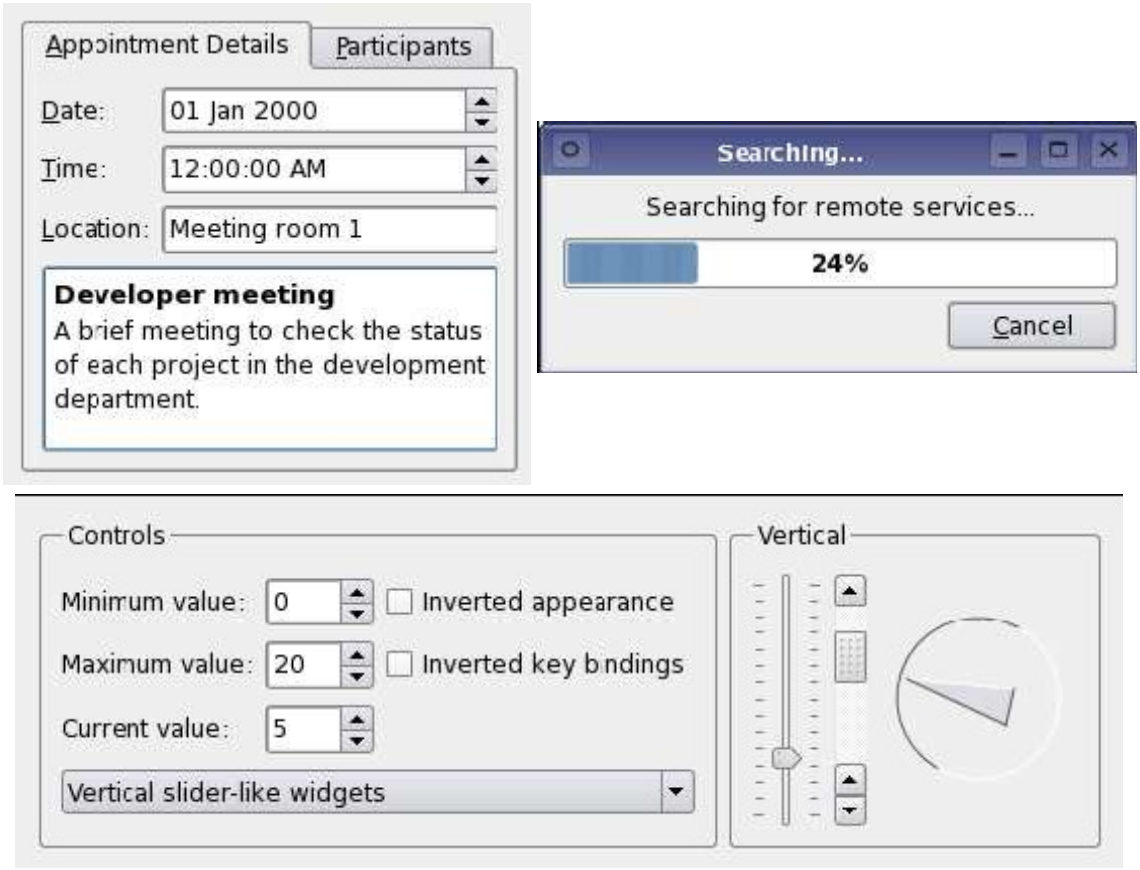


Figura 3.1: Exemples de widgets fets amb Qt.

3.1.3 Signals i slots

Les aplicacions amb interfície gràfica d'usuari responen a les accions de l'usuari. Per exemple, quan un usuari fa clic sobre un ítem del menú o un botó, l'aplicació executa un tros de codi. Més generalment, és útil poder comunicar objectes de qualsevol tipus entre ells, relacionar un determinat esdeveniment amb un codi concret.

El sistema que proporcionen les Qt per fer això s'anomena mecanisme de *"signals i slots"*. Els *signals i slots* són flexibles, orientats a objectes i implementats amb C++.

Els *widgets* de Qt emeten *signals* quan es produeixen esdeveniments. Per exemple, si un usuari fa clic sobre un botó el botó emet un *signal* "clicked". Aquest *signal* es pot connectar a un *slot* (que és una funció) d'un altre objecte fent servir la funció **connect()**. Aquest mecanisme permet connectar objectes de classes totalment independents, sense coneixement l'una de l'altra, i d'aquesta manera ajuda a crear classes reutilitzables. A la Figura 3.2 es pot veure un esquema de connexions de *signals i slots*.

Com a exemple de funcionament, si volem fer que quan l'usuari faci clic sobre un botó es tanqui l'aplicació, es faria amb un codi com el següent:

```
connect(button, SIGNAL(clicked()),
qApp, SLOT(quit()));
```

Es poden fer i desfer connexions entre *signals* i *slots* en qualsevol moment durant l'execució d'una aplicació de Qt.

Perquè una classe pugui fer servir el mecanisme de *signals i slots* ha d'heredar de **QObject** o una de les seves subclasses i incloure la *macro* **`Q_OBJECT`** a la definició de la classe. Els *signals* es declaren a la secció `signals` i els *slots* a les seccions `public slots`, `protected slots` o `private slots`. Exemple:

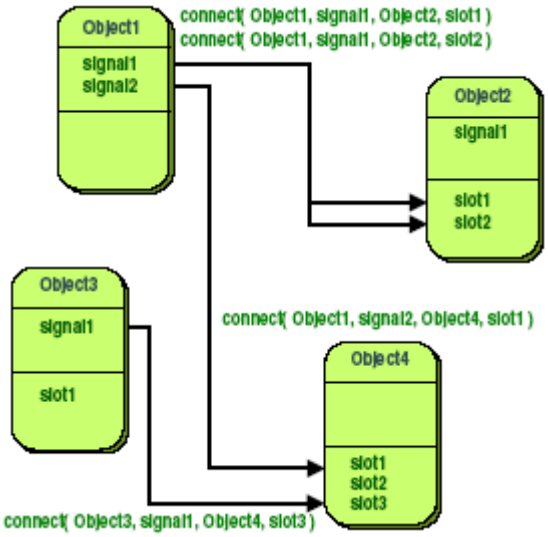


Figura 3.2: Vista abstracta d'algunes connexions de signals *i* slots.

```
class BankAccount : public QObject
{
    Q_OBJECT

public:
    BankAccount() { curBalance = 0; }
    int balance() const { return curBalance; }

public slots:
    void setBalance(int newBalance);

signals:
    void balanceChanged(int newBalance);

private:
    int currentBalance;
};
```

Quan s'emet un *signal* s'executen tots els *slots* que estan connectats amb ell. Els *slots* són com la resta dels mètodes de la classe, però amb la propietat que poden ser connectats a *signals*.

Implementació de setBalance():

```
void BankAccount::setBalance(int newBalance)
{
    if (newBalance != currentBalance) {
        currentBalance = newBalance;
        emit balanceChanged(currentBalance);
    }
}
```

Els *signals* s'emeten fent servir la paraula `emit` seguida del nom del *signal* amb els paràmetres corresponents. A diferència dels *slots*, els *signals* no els implementa l'usuari.

Per connectar dos BankAccounts diferents es faria així:

```
BankAccount x, y;
connect(&x, SIGNAL(balanceChanged(int)), &y, SLOT(setBalance(int)));
```

Quan es fa la connexió només s'indiquen els tipus dels paràmetres, però no els noms. També és possible connectar un *signal* a un altre *signal* directament, i quan s'emet el primer s'emet el segon automàticament. Perquè la connexió funcioni el *signal* i l'*slot* (o els dos *signals*) han de tenir la mateixa signatura pel que fa als tipus, encara que el receptor pot tenir menys paràmetres i d'aquesta manera ignoraria els altres. Si no és així, donarà un avís en temps d'execució perquè no podrà fer la connexió. Passaria el mateix si el *signal* o l'*slot* no existissin.

Per compilar les classes que fan servir el mecanisme de *signals i slots* aquestes han de passar pel *Meta-Object Compiler* (`moc`), inclòs amb les Qt, que converteix el codi dels *signals i slots* en codi C++ estàndard. El `moc` s'invoca automàticament amb els Makefiles que genera el `qmake` (una altra aplicació de les Qt).

3.1.4 Qt Designer

El Qt Designer (Figura 3.3) és una aplicació de les Qt que permet dissenyar ràpidament una interfície gràfica amb Qt. És senzill i intuïtiu. Només cal buscar el *widget* que volem entre els menús o les barres d'eines, i llavors afegir-lo al lloc que vulguem del *widget* pare. Després es poden canviar les propietats del *widget* amb l'editor de propietats. També es poden aplicar diferents tipus de *layouts* (organitzacions dels *widgets*).

El Qt Designer té plantilles per diferents tipus de GUIs: finestres principals, quadres de diàleg, etc. També es poden crear noves plantilles i afegir *widgets* personalitzats per fer-los servir en altres interfícies gràfiques.

Un cop dissenyada la interfície es pot veure com queda amb diferents estils sense necessitat de compilar. D'aquesta manera s'estalvia molt temps si s'han de corregir petits detalls.

Amb el Qt Designer també es poden fer les connexions entre *signals* i *slots* dels diferents *widgets* de la interfície, i afegir *signals*, *slots*, atributs i mètodes a la pròpia interfície.

Les interfícies d'usuari creades amb el Qt Designer es guarden en fitxers amb extensió `.ui` que conté un codi XML que serveix per generar el codi font de la classe. Això es fa amb el programa `uic` (*User Interface Compiler*) de les Qt.

Qt permet agrupar les classes i interfícies gràfiques en projectes de Qt, que es guarden en fitxers `.pro`. El `qmake` genera automàticament els Makefiles necessaris per compilar els projectes de Qt, cridant els diferents programes (`qmake`, `uic`, `moc`) quan és necessari.

3.1.5 Ús en el projecte

Les Qt han servit per dissenyar la part d'interfície gràfica del projecte. La interfície que té els controls per configurar els paràmetres de visualització ha estat dissenyada amb el Qt Designer. S'ha aprofitat el mecanisme de signals i slots ha servit per construir classes amb un baix acoblament, tant les classes

de la interfície com les classes de control, implementant moltes de les comunicacions entre elles mitjançant aquest mecanisme.

Les Qt també han estat imprescindibles per poder integrar aquest projecte a la plataforma de visualització d'imatges mèdiques.

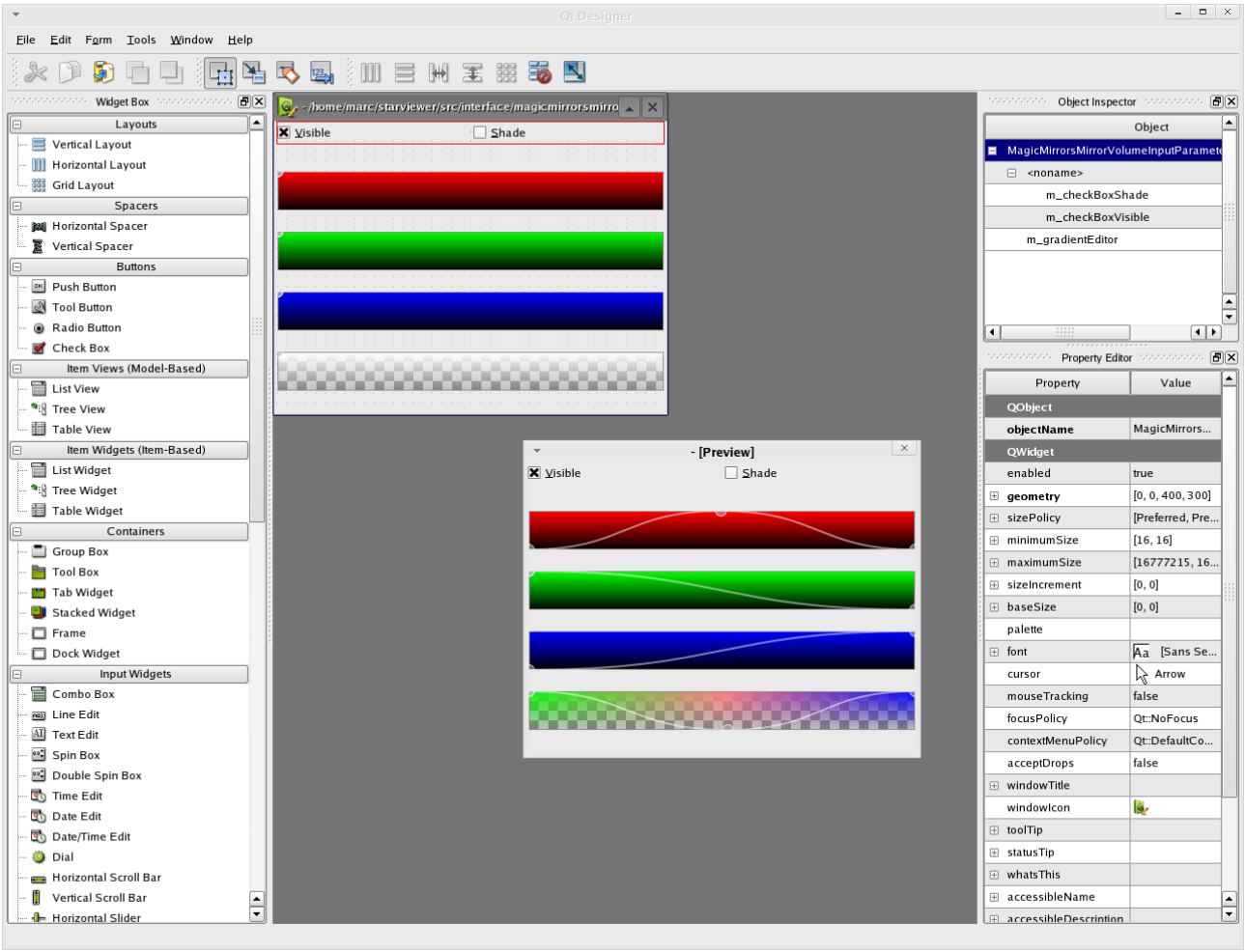


Figura 3.3: Qt Designer.

3.2 ITK

La informació d'aquest apartat està treta del lloc web d'ITK [10], on també hi ha la documentació i les biblioteques.

3.2.1 Descripció

ITK (*Insight ToolKit*) són unes biblioteques de codi obert per realitzar operacions de registre i segmentació d'imatges. La segmentació és el procés d'identificar i classificar les dades que es troben en una representació mostrejada digitalment. Normalment aquesta representació és una imatge obtinguda amb instruments mèdics com escànners de CT o MRI. Una operació de registre consisteix a

alinear o desenvolupar correspondències entre dades. Per exemple, en l'entorn mèdic, es poden alinear les dades d'una CT amb les d'una MRI per combinar la informació d'ambdues.

Les ITK estan implementades amb C++ fent servir programació genèrica (*templates*) i són multiplataforma.

Algunes característiques:

- Proporcionen representació de dades i algorismes per efectuar operacions de segmentació i registre. Estan enfocades cap a aplicacions mèdiques, encara que poden processar altres tipus de dades.
- Proporcionen representacions de dades de forma general per imatges de dimensions arbitràries i malles no estructurades.
- Suporten processament paral·lel amb *multi-threading* i memòria compartida.
- Estan organitzades al voltant d'una arquitectura de flux de dades. Les dades es representen amb objectes de dades que són processats per objectes de procés (filtres). Els objectes de dades i els objectes de procés es connecten en *pipelines*. Les *pipelines* són capaces de processar les dades a trossos segons el límit de memòria que especifiqui l'usuari.
- Els objectes s'instancien utilitzant factories d'objectes (patró de disseny *Factory Method*).
- Els esdeveniments es processen fent servir el patró de disseny *Observer*.
- Les llibreries estan implementades seguint els principis de la programació genèrica (*templates* de C++).
- Són multiplataforma (Unix, Windows i Mac OS X).
- El model de memòria depèn dels *"smart pointers"*, que tenen un comptador de referències a objectes.

3.2.2 Ús en el projecte

Les llibreries ITK no les he fetes servir directament en cap dels meus mòduls, però sí que es fan servir en altres parts de la plataforma de les quals en depenen els meus mòduls.

3.3 VTK

La informació d'aquest apartat està treta del lloc web de VTK [11], on també hi ha la documentació i les biblioteques.

3.3.1 Descripció

Les VTK (*Visualization ToolKit*) són unes biblioteques de codi obert gratuïtes per a gràfics per computador, processament d'imatges i visualització. Estan implementades amb C++ (orientades a objectes) i són multiplataforma.

Els model de gràfics de VTK està a un nivell d'abstracció més alt que altres llibreries gràfiques com OpenGL o PEX, i per tant es poden crear aplicacions gràfiques més fàcilment i ràpida.

És un sistema de visualització complet amb molts algorismes de visualització geomètrics i volumètrics, a més de nombrosos filtres per tractar les imatges abans de visualitzar-les. També és possible barrejar imatges 2D, 3D i dades.

Algunes característiques:

- Implementat amb C++ i orientat a objectes.
- Es pot integrar amb varis sistemes de finestres, entre ells Qt.
- Les finestres de VTK es poden fer interactives.
- Tracta els esdeveniments seguint el patró *Observer*.
- Visualització de superfícies i volums amb diferents algoritmes, entre ells *ray casting*.
- Característiques de sistemes de visualització: llums, càmeres, textures, etc.
- El codi per fer la visualització és independent del dispositiu de visualització.
- Gran quantitat de filtres per tractar les dades.
- Sistema de visualització amb *pipelines* (flux de dades) que pot tractar les dades per parts.
- Execució paral·lela de *pipelines* i filtres (*multi-threading*).
- Comptadors de referències pels objectes de VTK.
- Creació d'objectes seguint el patró *Factory Method*.

Les VTK estan formades per dos models d'objectes: el model de gràfics i el model de visualització.

3.3.2 El model de gràfics

L'objectiu del model de gràfics és transformar dades gràfiques en imatges.

El model de gràfics inclou els actors i volums (*props*), llums, càmeres, propietats i *mappers* dels *props*, transformacions, LUTs (*Look-Up Tables*) i funcions de transferència, *renderers*, *render windows* i *render window interactors*.

Combinant aquests elements es crea una escena.

Els *props* (actors i volums) són els objectes que es veuen a l'escena. La seva aparença és controlada per una propietat (fent servir LUTs o funcions de transferència) i la seva geometria per un *mapper* (que proporciona una interfície entre el model de visualització i el model de gràfics). Els *props* també guarden una transformació interna.

Les càmeres i les llums funcionen de forma semblant a altres sistemes de visualització, però a VTK són objectes.

El *renderer* és l'objecte que dibuixa la imatge a la *render window*, que és la finestra que mostra l'escena. El *render window interactor* serveix perquè l'usuari pugui interaccionar amb l'escena a través de la finestra.

3.3.3 El model de visualització

L'objectiu del model de visualització és transformar informació en dades gràfiques.

Aquest model inclou dos tipus d'objectes: els objectes de dades i els objectes de procés.

Els objectes de dades representen dades de diferents tipus, amb estructura o sense. Les imatges de VTK són un tipus de dades.

Els objectes de procés o filtres processen objectes de dades d'entrada per produir nous objectes de dades de sortida. Representen els algorismes del sistema.

Els dos tipus d'objectes es connecten per formar *pipelines* de visualització, que s'executen només quan es demanen les dades.

3.3.4 Ús en el projecte

Les VTK han servit per implementar tota la part de visualització: *ray casting*, funcions de transferència, volums, miralls, plans, càmeres, etc.

4 Anàlisi i visió general de l'aplicació

En aquest capítol farem l'anàlisi de requeriments i el disseny de l'aplicació. Veurem primer de tot una descripció dels requeriments de l'aplicació, tant els funcionals com els no funcionals. Tot seguit veurem el diagrama de casos d'ús i la fitxa de cada cas. Finalment donarem una visió general de l'aplicació, tot explicant els mòduls que hi haurà, per entrar en detall en el disseny en el capítol següent.

4.1 Anàlisi de requeriments

Com ja hem dit a la introducció, l'aplicació desenvolupada s'ha d'integrar en una plataforma de visualització d'imatges mèdiques, que en el moment de l'anàlisi permetia obrir un o més models de vòxels. Per tant en l'anàlisi hem suposat que podem accedir a un conjunt de models de vòxels carregats a memòria a través els mètodes que proporciona la plataforma.

4.1.1 Requeriments funcionals

Separarem els requeriments funcionals en tres grups, segons els tres problemes que hem de resoldre.

Visualització de models fusionats

- Seleccionar el/s volum/s a visualitzar.
- Triar el nombre de miralls.
- Definir la/les propietat/s que es veu/en al centre i a cada mirall.
- Definir la funció de transferència de cada propietat al centre i a cada mirall.
- Moure els miralls a qualsevol posició.
- Manipular el model i la vista principal lliurement.
- Els miralls han de visualitzar la projecció del model en el punt on es trobin.

Selecció del punt de vista

- Seleccionar el volum per al qual s'ha d'optimitzar el punt de vista.
- Definir els paràmetres de la segmentació per crear el model simplificat.
- Segmentar el volum per simplificar-lo.
- Triar el nombre de plans que es faran servir.

- Definir els paràmetres de càlcul de l'*excess entropy* per trobar el millor punt de vista (incloent el nombre de raigs llançats i les mostres que es prenen per raig, entre d'altres).
- Definir la funció de transferència de la visualització.
- Manipular el model i la vista principal lliurement.
- Cada pla ha de tenir una finestra amb la seva visualització.
- Calcular l'*excess entropy* de la vista de cada pla.
- L'aplicació mostrarà els resultats.

Assignació de colors

- Definir cada component del color i l'opacitat per cada valor de propietat.
- Calcular una funció de transferència que s'ajusti al model.
- Modificar la funció de transferència calculada.

4.1.2 Requeriments no funcionals

Els requeriments no funcionals són més generals, a nivell global de l'aplicació.

Per començar, caldrà que la interfície d'usuari sigui tan intuïtiva com sigui possible i estigui ben integrada a la plataforma. Òbviament, cal que permeti a l'usuari accedir a totes les funcionalitats.

Per tal de complir el requeriment de la integració amb la plataforma, cal que l'aplicació faci servir les mateixes biblioteques i el mateix entorn de treball. Per tant caldrà desenvolupar l'aplicació sota un sistema operatiu Linux. El compilador serà el GCC 4.x. També caldran les biblioteques Qt 4.1, ITK 2.6.0 i VTK 5.0.0. Amb això es compleix el requeriment que l'aplicació es desenvolupi amb programari lliure.

4.2 Casos d'ús

En aquest capítol explicarem els casos d'ús per a cadascun dels tres problemes a resoldre. Per cada problema hi haurà un diagrama de casos d'ús i una fitxa per cada cas d'ús.

4.2.1 Visualització de models fusionats

Diagrama de casos d'ús

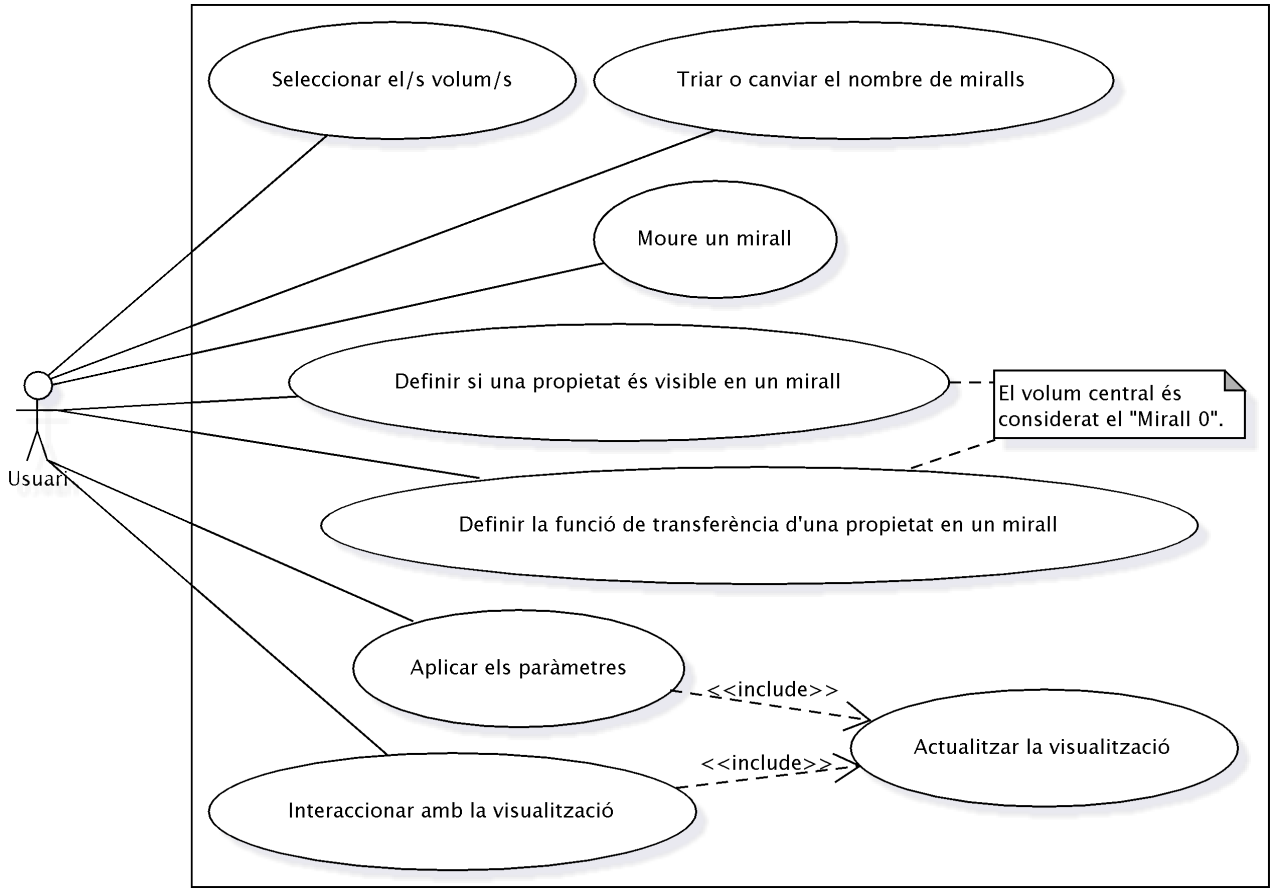


Figura 4.1: Diagrama de casos d'ús per a la visualització de models fusionats.

Fitxes de casos d'ús

Cas d'ús	**Seleccionar el/s volum/s**
Descripció	L'usuari tria un o més volums d'entre tots els que hi ha oberts per visualitzar-lo/s.
Actors	Usuari
Precondició	L'usuari encara no ha seleccionat cap volum.
Flux principal	1. El sistema obté una llista de volums oberts. 2. Mostra la llista a l'usuari. 3. L'usuari tria els que vol visualitzar. 4. El sistema guarda la llista de volums seleccionats i prepara la interfície per definir els paràmetres de visualització.

Cas d'ús	**Seleccionar el/s volum/s**
Flux alternatiu	L'usuari pot cancel·lar l'acció o acceptar sense haver seleccionat cap volum. En qualsevol dels dos casos no s'executa el pas 4.
Postcondició	El/s volum/s ha/n estat seleccionat/s i ja es poden definir els seus paràmetres de visualització.
Comentaris	Aquesta acció s'ha d'executar preferentment al principi, i forçosament abans de qualsevol cas d'ús que tracti amb volums. Només es pot executar una vegada seguint el flux principal. Si l'usuari selecciona un sol volum es considera un model simple. Si en selecciona més d'un es considera un model fusionat. En el cas de models fusionats, s'ha d'haver executat l'operació de registre prèviament, amb un altre mòdul de la plataforma. Aquí es considerarà que l'operació de registre s'ha realitzat, però no es comprovarà.

Cas d'ús	**Triar o canviar el nombre de miralls**
Descripció	L'usuari tria el nombre de miralls que vol en un moment donat.
Actors	Usuari
Precondició	
Flux principal	1. L'usuari indica al sistema quants miralls vol. 2. El sistema prepara la interfície per definir tots els paràmetres relacionats amb els miralls d'acord amb la nova quantitat de miralls, afegint opcions de configuració per als nous miralls que s'hagin afegit o traient les opcions de configuració dels miralls esborrats.
Flux alternatiu	
Postcondició	La interfície permet configurar el nombre exacte de miralls que vol l'usuari.
Comentaris	Els canvis no es reflectiran a la visualització fins que s'apliquin els paràmetres.

Cas d'ús	**Moure un mirall**
Descripció	L'usuari canvia la posició d'un mirall determinat. L'orientació sempre és cap al centre.
Actors	Usuari
Precondició	
Flux principal	1. L'usuari selecciona el mirall. 2. Defineix la distància des del centre. 3. Defineix la latitud (en graus, positiu = nord, negatiu = sud). 4. Defineix la longitud (en graus, positiu = est, negatiu = oest).
Flux alternatiu	
Postcondició	La nova posició del mirall ha quedat definida.
Comentaris	El punt de latitud 0º i longitud 0º es troba a l'eix Z. Els canvis no es reflectiran a la visualització fins que s'apliquin els paràmetres.

Cas d'ús	**Definir si una propietat és visible en un mirall**
Descripció	L'usuari defineix si una propietat determinada del model fusionat és visible o no en un mirall determinat.
Actors	Usuari
Precondició	L'usuari ha seleccionat un o més volums.
Flux principal	1. L'usuari selecciona el mirall. 2. Selecciona la propietat (el volum). 3. Defineix si és visible o no.
Flux alternatiu	
Postcondició	L'estat de visibilitat de la propietat seleccionada al mirall seleccionat ha quedat definit.
Comentaris	Tot i que aquesta utilitat està pensada per a models fusionats també es pot fer servir en un model simple. Els canvis no es reflectiran a la visualització fins que s'apliquin els paràmetres.

Cas d'ús	**Definir la funció de transferència d'una propietat en un mirall**
Descripció	L'usuari defineix la funció de transferència (color i opacitat) d'una propietat determinada en un mirall determinat.
Actors	Usuari
Precondició	L'usuari ha seleccionat un o més volums.
Flux principal	1. L'usuari selecciona el mirall. 2. Selecciona la propietat (el volum). 3. Defineix la funció de transferència.
Flux alternatiu	
Postcondició	La funció de transferència de la propietat seleccionada al mirall seleccionat ha quedat definida.
Comentaris	Els detalls de com es defineix la funció de transferència formen part d'un altre cas d'ús. Els canvis no es reflectiran a la visualització fins que s'apliquin els paràmetres.

Cas d'ús	**Aplicar els paràmetres**
Descripció	S'apliquen tots els paràmetres definits en altres casos d'ús.
Actors	Usuari
Precondició	L'usuari ha seleccionat un o més volums.
Flux principal	1. L'usuari activa l'acció d'aplicar els paràmetres. 2. El sistema aplica els nous paràmetres a les classes que fan la visualització. 3. Actualitza la visualització.
Flux alternatiu	

Cas d'ús	**Aplicar els paràmetres**
Postcondició	Els paràmetres han estat aplicats i la visualització s'ha actualitzat d'acord amb els nous paràmetres.
Comentaris	Els detalls de com s'actualitza la visualització formen part d'un altre cas d'ús.

Cas d'ús	**Interaccionar amb la visualització**
Descripció	L'usuari interacciona amb la finestra de la visualització per controlar la càmera o moure o girar el model.
Actors	Usuari
Precondició	Existeix la finestra de visualització.
Flux principal	1. L'usuari realitza la interacció. 2. El sistema actualitza la visualització.
Flux alternatiu	
Postcondició	La visualització està actualitzada d'acord amb els canvis produïts per la interacció de l'usuari.
Comentaris	Els detalls de com s'actualitza la visualització formen part d'un altre cas d'ús.

Cas d'ús	**Actualitzar la visualització**
Descripció	S'actualitza la visualització dels Miralls Màgics.
Actors	Sistema
Precondició	S'han aplicat els paràmetres.
Flux principal	1. El sistema, per cada mirall: A. Fa la visualització des del punt de vista del mirall. B. Converteix la imatge visualitzada en una textura. C. Enganxa la textura al pla que representa el mirall. 2. Si és la primera vegada, crea la finestra de visualització. 3. Fa la visualització principal del model amb els miralls.
Flux alternatiu	
Postcondició	La visualització està actualitzada.
Comentaris	El pas 3 és l'únic que es fa sempre. El pas 1 només es fa quan s'apliquen els paràmetres i quan l'usuari interacciona amb el model (quan interacciona amb la càmera no).

4.2.2 Selecció del punt de vista

Diagrama de casos d'ús

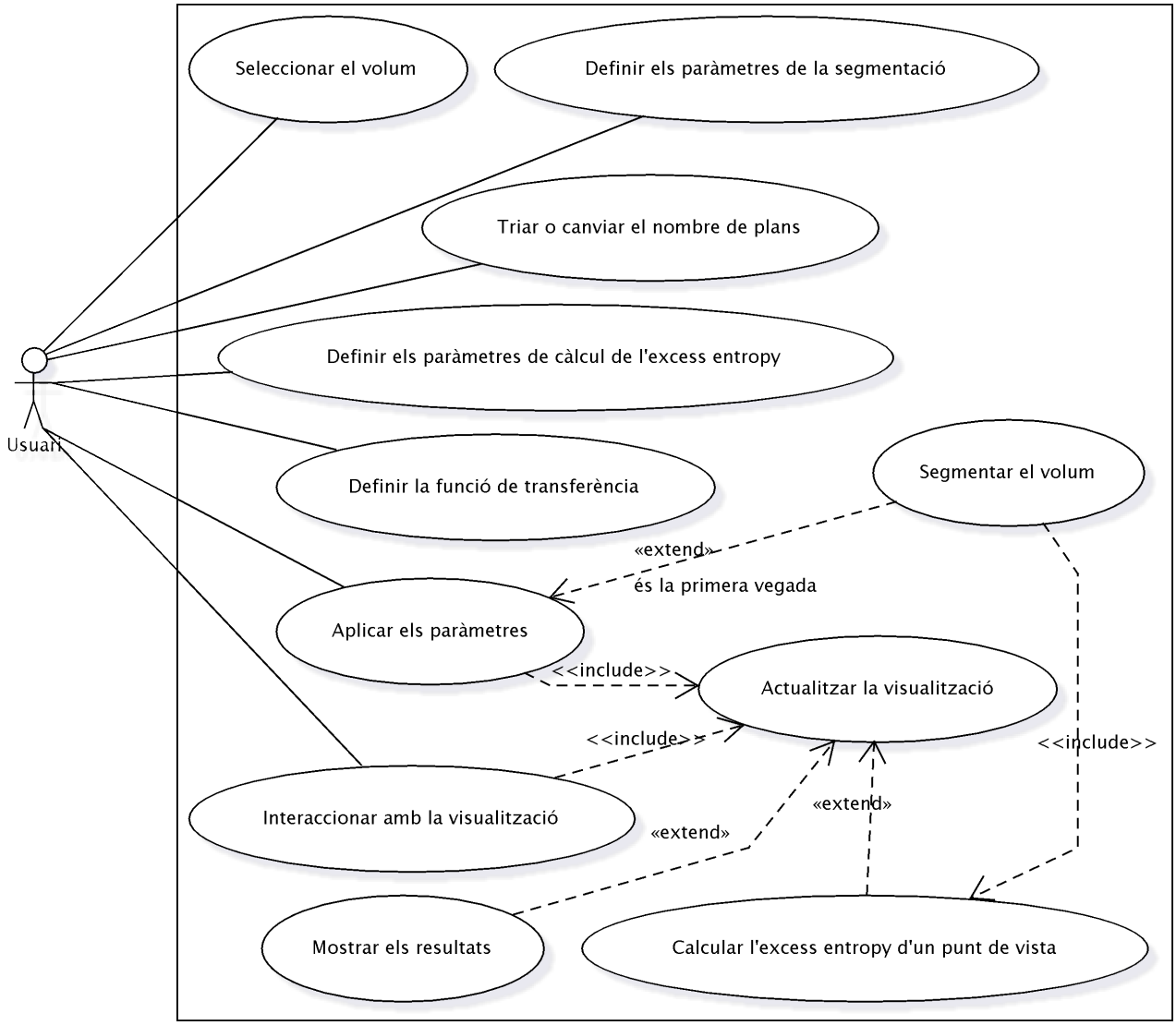


Figura 4.2: Diagrama de casos d'ús per a la selecció del punt de vista.

Fitxes de casos d'ús

Cas d'ús	**Seleccionar el volum**
Descripció	L'usuari tria un volum d'entre tots els que hi ha oberts per trobar el seu millor punt de vista.
Actors	Usuari
Precondició	L'usuari encara no ha seleccionat cap volum.

Cas d'ús	**Seleccionar el volum**
Flux principal	1. El sistema obté una llista de volums oberts. 2. Mostra la llista a l'usuari. 3. L'usuari tria el que vol visualitzar. 4. El sistema guarda el volum seleccionat.
Flux alternatiu	L'usuari pot cancel·lar l'acció o acceptar sense haver seleccionat cap volum. En qualsevol dels dos casos no s'executa el pas 4.
Postcondició	El volum ha estat seleccionat i l'usuari ja pot aplicar els paràmetres quan vulgui.
Comentaris	Només es pot executar una vegada seguint el flux principal.

Cas d'ús	**Definir els paràmetres de la segmentació**
Descripció	L'usuari defineix els paràmetres de la segmentació.
Actors	Usuari
Precondició	No s'ha segmentat el volum (no s'han aplicat els paràmetres cap vegada).
Flux principal	1. L'usuari defineix els paràmetres de la segmentació, incloent: ○ Nombre de raigs. ○ Nombre de mostres per raig. ○ Longitud de bloc. ○ Nombre de clústers.
Flux alternatiu	
Postcondició	Els paràmetres de la segmentació han quedat definits.
Comentaris	Tots aquests paràmetres serveixen per calcular l'*excess entropy*, que serà necessari durant la segmentació. El nombre de clústers a més a més és un paràmetre de l'algorisme de segmentació. Hi ha pot haver altres paràmetres addicionals depenent de l'algorisme de segmentació. La segmentació es farà quan s'apliquin els paràmetres per primer cop.

Cas d'ús	**Triar o canviar el nombre de plans**
Descripció	L'usuari tria el nombre de plans que vol fer servir per trobar el millor punt de vista.
Actors	Usuari
Precondició	
Flux principal	1. L'usuari tria quants plans vol, escollint entre 4, 6, 8, 12 i 20.
Flux alternatiu	
Postcondició	El nombre de plans ha quedat definit.
Comentaris	Els canvis no es reflectiran a la visualització fins que s'apliquin els paràmetres.

Cas d'ús	**Definir els paràmetres de càlcul de l'*excess entropy***
Descripció	L'usuari defineix els paràmetres de càlcul de l'*excess entropy* d'un punt de vista (els mateixos paràmetres són per tots els punts de vista).
Actors	Usuari
Precondició	
Flux principal	1. L'usuari defineix els paràmetres de càlcul de l'*excess entropy* d'un punt de vista, incloent: ○ Nombre de raigs. ○ Nombre de mostres per raig. ○ Longitud de bloc.
Flux alternatiu	
Postcondició	Els paràmetres de càlcul de l'*excess entropy* d'un punt de vista han quedat definits.
Comentaris	El nombre de clústers, que també és necessari per calcular l'*excess entropy*, s'agafa directament dels paràmetres de la segmentació perquè ha de ser igual. Els altres paràmetres poden ser diferents dels que s'han definit per a la segmentació. L'*excess entropy* no es tornarà a calcular fins que s'apliquin els paràmetres.

Cas d'ús	**Definir la funció de transferència**
Descripció	L'usuari defineix la funció de transferència (color i opacitat) amb la qual es visualitzarà el volum a tot arreu.
Actors	Usuari
Precondició	
Flux principal	1. L'usuari defineix la funció de transferència.
Flux alternatiu	
Postcondició	La funció de transferència ha quedat definida.
Comentaris	Els detalls de com es defineix la funció de transferència formen part d'un altre cas d'ús. Els canvis no es reflectiran a la visualització fins que s'apliquin els paràmetres.

Cas d'ús	**Aplicar els paràmetres**
Descripció	S'apliquen tots els paràmetres definits en altres casos d'ús.
Actors	Usuari
Precondició	L'usuari ha seleccionat un volum.
Flux principal	1. L'usuari activa l'acció d'aplicar els paràmetres. 2. El sistema aplica els nous paràmetres a les classes que fan la visualització i els càlculs. 3. Si és la primera vegada que s'apliquen els paràmetres, segmenta el volum. 4. Actualitza la visualització.
Flux alternatiu	

Cas d'ús	**Aplicar els paràmetres**
Postcondició	Els paràmetres han estat aplicats, el volum ja s'ha segmentat i la visualització s'ha actualitzat d'acord amb els nous paràmetres.
Comentaris	Els detalls de com es segmenta el volum i com s'actualitza la visualització formen part d'altres casos d'ús.

Cas d'ús	**Interaccionar amb la visualització**
Descripció	L'usuari interacciona amb la finestra de la visualització per controlar la càmera o moure o girar el model.
Actors	Usuari
Precondició	Existeix la finestra de visualització.
Flux principal	1. L'usuari realitza la interacció. 2. El sistema actualitza la visualització.
Flux alternatiu	
Postcondició	La visualització està actualitzada d'acord amb els canvis produïts per la interacció de l'usuari.
Comentaris	Els detalls de com s'actualitza la visualització formen part d'un altre cas d'ús.

Cas d'ús	**Segmentar el volum**
Descripció	El sistema aplica un algorisme de segmentació sobre el volum per construir un model simplificat, apte per calcular l'*excess entropy*.
Actors	Sistema
Precondició	S'han aplicat els paràmetres i el volum no s'ha segmentat.
Flux principal	1. El sistema aplica un algorisme de segmentació, que consistirà en realitzar reiteradament les accions següents: A. Fer una segmentació concreta amb el nombre de clústers definits per l'usuari. B. Calcular l'*excess entropy* des d'un punt de vista fix del model segmentat. C. Si l'*excess entropy* calculada és més gran que la màxima fins ara, aquesta segmentació passa a ser la millor i l'*excess entropy* calculada passa a ser la màxima. 2. El model resultat de la millor segmentació es converteix en el model simplificat.
Flux alternatiu	
Postcondició	S'ha creat el model simplificat.
Comentaris	Només es fa una vegada, quan s'apliquen els paràmetres per primer cop. Els detalls de com es calcula l'*excess entropy* formen part d'un altre cas d'ús.

Cas d'ús	**Actualitzar la visualització**
Descripció	S'actualitza la visualització principal.
Actors	Sistema
Precondició	S'han aplicat els paràmetres.
Flux principal	1. El sistema, per cada pla: A. Fa la visualització des del punt de vista del pla, creant la finestra de visualització si encara no existia. I. Durant la visualització calcula l'*excess entropy*, si cal. B. Converteix la imatge visualitzada en una textura. C. Enganxa la textura al pla que representa el mirall. 2. Si és la primera vegada, crea la finestra de visualització. 3. Fa la visualització principal del model amb els plans al voltant. 4. Si s'ha executat el pas I, es mostren els resultats.
Flux alternatiu	
Postcondició	La visualització està actualitzada i l'*excess entropy* de cada punt de vista també.
Comentaris	El pas 3 és l'únic que es fa sempre. El pas 1 només es fa quan s'apliquen els paràmetres i quan l'usuari interacciona amb el model (quan interacciona amb la càmera no). El sistema decideix que cal calcular l'*excess entropy* quan han canviat els paràmetres per calcular-la o quan l'usuari ha interaccionat amb el model. Els detalls de com es calcula l'*excess entropy* i com es mostren els resultats formen part d'altres casos d'ús.

Cas d'ús	**Calcular l'*excess entropy* d'un punt de vista**
Descripció	El sistema calcula l'*excess entropy* d'un punt de vista determinat.
Actors	Sistema
Precondició	S'han aplicat els paràmetres.
Flux principal	1. Donat un punt de vista, el sistema llança el nombre de raigs definit per l'usuari. Per cada raig: A. Agafa el nombre de mostres definit per l'usuari. Per cada mostra: I. Es guarda la mostra en una llista de les L últimes mostres, on L és la longitud de bloc definida per l'usuari. 2. Va construint un histograma de mostres de longitud L i de longitud $L - 1$ (una mostra de longitud L consisteix en la llista ordenada amb L mostres). 3. Quan ha acabat de construir els histogrames, calcula l'entropia dels blocs de longitud L i de longitud $L - 1$, a partir dels histogrames. 4. Calcula l'*entropy rate* a partir de l'entropia dels blocs de longitud L i de longitud $L - 1$. 5. Calcula l'*excess entropy* a partir de l'entropia dels blocs de longitud L i l'*entropy rate*. 6. Guarda el resultat del càlcul de l'*excess entropy*.
Flux alternatiu	
Postcondició	L'*excess entropy* d'aquest punt de vista està actualitzada.

Cas d'ús	**Calcular l'*excess entropy* d'un punt de vista**
Comentaris	El resultat queda guardat i pot ser consultat en qualsevol moment.

Cas d'ús	**Mostrar els resultats**
Descripció	El sistema mostra els resultats del càlcul de l'*excess entropy* de tots els punts de vista.
Actors	Sistema
Precondició	S'ha calculat l'*excess entropy* de tots els punts de vista.
Flux principal	1. El sistema consulta els resultats de tots els punts de vista. 2. Mostra tots els resultats a l'usuari.
Flux alternatiu	
Postcondició	Els resultats han estat mostrats a l'usuari.
Comentaris	Només s'executarà quan s'hagi calculat l'*excess entropy* de tots els plans.

4.2.3 Assignació de colors

Diagrama de casos d'ús

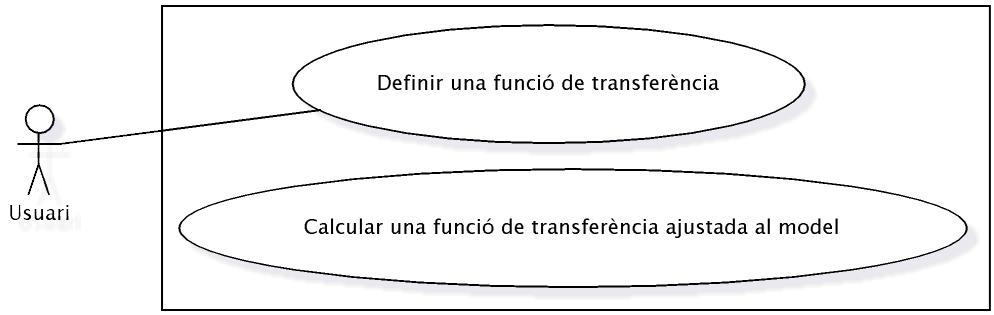


Figura 4.3: Diagrama de casos d'ús per a l'assignació de colors.

Fitxes de casos d'ús

Cas d'ús	**Definir una funció de transferència**
Descripció	L'usuari defineix una nova funció de transferència o en modifica una d'existent. Això consistirà en establir una correspondència entre valors de propietat i colors RGBA (color i opacitat).
Actors	Usuari
Precondició	

Cas d'ús	**Definir una funció de transferència**
Flux principal	1. L'usuari defineix la funció de transferència creant punts de correspondència, on cada punt estableix la correspondència entre un valor de propietat i un color RGBA. L'usuari en tot moment pot: ○ Crear un nou punt. ○ Modificar un punt existent. ○ Esborrar un punt existent.
Flux alternatiu	
Postcondició	La funció de transferència ha quedat definida.
Comentaris	La manera com es defineixen els punts es decidirà en el disseny. En els trams entre cada parell de punts consecutius la funció de transferència queda definida per interpolació.

Cas d'ús	**Calcular una funció de transferència ajustada al model**
Descripció	El sistema defineix una funció de transferència que s'ajusta a un model simple. Això consistirà en establir una correspondència entre valors de propietat i colors RGBA (color i opacitat). Aquesta funció de transferència es podrà definir després d'haver segmentat el model.
Actors	Sistema
Precondició	S'ha segmentat un model simple.
Flux principal	1. El sistema obté els límits de la segmentació (els valors que defineixen les fronteres entre els diferents clústers). 2. Crea una funció de transferència amb tants trams o intervals com clústers hi ha en el model segmentat. Cada interval és entre dos límits consecutius i té un color RGBA diferent dels altres trams.
Flux alternatiu	
Postcondició	S'ha definit una funció de transferència ajustada al model.
Comentaris	En els trams entre cada parell de punts consecutius la funció de transferència queda definida per interpolació. La funció de transferència pintarà cada clúster amb un sol color homogeni i diferent dels altres.

4.3 Visió general de l'aplicació

L'aplicació que es desenvoluparà en aquest projecte s'haurà d'integrar en la plataforma StarViewer de visualització i processament de dades mèdiques. Aquest fet ens marca les pautes a seguir en el moment de dissenyar i implementar els mòduls necessaris per poder assolir els nostres objectius.

Hem de tenir en compte que l'aplicació està composta, des del punt de vista lògic, de diversos mòduls, cadascun dels quals té un conjunt de funcions relacionades entre elles. Cada mòdul es pot subdividir en diferents submòduls, per realitzar tasques més concretes, i es poden anar fent subdivisions fins arribar al nivell de les classes individuals.

L'organització real de les classes, però, és en *packages* o paquets clarament definits, que són els que existeixen a la plataforma. Aquests paquets separen les classes segons siguin de la interfície, eines, algorismes de registre, algorismes de segmentació, algorismes de visualització, etc. A l'apartat del disseny de la plataforma, al capítol següent, veurem amb més detall els paquets que hi ha i quins tipus de classes els corresponen.

Degut a aquesta diferència entre l'organització lògica i la física, cada mòdul s'estén entre uns quants paquets. Tot seguit explicarem els mòduls que hi ha i els submòduls que té cadascun a cada paquet.

4.3.1 Mòdul de la plataforma

Aquest mòdul fa referència a totes les classes existents a la plataforma en la seva versió bàsica, abans d'incorporar-hi les classes creades en aquest projecte. Les classes més importants d'aquest mòdul es troben a als paquets interface i repositories, i són les següents:

- A interface: Director, Parameters, QApplicationMainWindow, QInputParameters, QworkingAreaWidget.
- A repositories: Identifier, Volume, VolumeRepository.

4.3.2 Mòdul dels Miralls Màgics

Aquest mòdul conté totes les classes que estan dissenyades específicament per implementar els Miralls Màgics, resolent el problema de la visualització de models fusionats. Aquest mòdul té submòduls als paquets interface i visualization. Podem veure el seu diagrama de classes a la Figura 5.4 (pàgina 58).

A interface:

- **Submòdul d'introducció de paràmetres:** està format per totes les classes que permeten introduir els paràmetres d'entrada dels Miralls Màgics. Conté les classes següents:
  - MagicMirrorsMirrorVolumeInputParametersFormBase
  - MagicMirrorsMirrorVolumeInputParametersForm
  - MagicMirrorsMirrorInputParametersFormBase
  - MagicMirrorsMirrorInputParametersForm
  - MagicMirrorsInputParametersFormBase
  - MagicMirrorsInputParametersForm
- **Submòdul d'encapsulament de paràmetres:** la seva funció és guardar tots els paràmetres d'entrada dels Miralls Màgics. Conté una sola classe: MagicMirrorsParameters.

- **Submòdul de la finestra de visualització principal:** com el seu nom indica, és on hi ha la finestra de visualització principal, on es veu el model amb els miralls al voltant. Només conté la classe MagicMirrorsViewer.

- **Submòdul d'enllaç de la interfície amb les classes de control:** aquest submòdul s'encarrega de relacionar les classes que formen part de la interfície amb les classes de control, les que fan la visualització, que es troben al paquet visualization. L'única classe que en forma part és MagicMirrorsDirector.

A visualization:

- **Submòdul del control principal de la visualització:** aquí hi ha les classes que controlen la visualització principal. Està format per les classes MagicMirrors i MagicMirrorsUpdater.

- **Submòdul de volums:** aquest submòdul s'encarrega de gestionar tot el que fa referència als volums: guardar els paràmetres de visualització de cada volum a cada mirall, mantenir sincronitzats els diversos volums que formen un model registrat, etc. Està format per dues classes: MagicMirrorsVolume i MagicMirrorsVolumeObserver.

- **Submòdul de miralls:** la seva funció és mantenir i controlar tot el que fa referència a la visualització i el control dels miralls. La seva única classe és MagicMirrorsMirror.

4.3.3 Mòdul de la selecció del punt de vista

Aquest mòdul conté totes les classes que estan dissenyades específicament per implementar la solució al problema de la selecció del punt de vista òptim. Aquest mòdul té submòduls als paquets interface i visualization. Podem veure el seu diagrama de classes a la Figura 5.16 (pàgina 72).

A interface:

- **Submòdul d'introducció de paràmetres:** està format per totes les classes que permeten introduir els paràmetres d'entrada d'aquest mòdul. Conté les classes següents:

  - OptimalViewpointInputParametersFormBase

  - OptimalViewpointInputParametersForm

- **Submòdul d'encapsulament de paràmetres:** la seva funció és guardar tots els paràmetres d'entrada d'aquest mòdul. Conté una sola classe: OptimalViewpoinParameters.

- **Submòdul de la finestra de visualització principal:** com el seu nom indica, és on hi ha la finestra de visualització principal, on es veu el model amb els plans al voltant. Només conté la classe OptimalViewpointViewer.

- **Submòdul d'enllaç de la interfície amb les classes de control:** aquest submòdul s'encarrega de relacionar les classes que formen part de la interfície amb les classes de control, les que fan la visualització, que es troben al paquet visualization. L'única classe que en forma part és OptimalViewpointDirector.

A visualization:

- **Submòdul del control principal de la visualització:** aquí hi ha les classes que controlen la visualització principal. Està format per les classes OptimalViewpoint i OptimalViewpointHelper.

- **Submòdul de volums:** aquest submòdul s'encarrega de gestionar tot el que fa referència als volums: guardar els paràmetres de visualització del volum, segmentar el volum per crear el model simplificat, etc. Està format per dues classes: OptimalViewpointVolume i vtkVolumeRayCastCompositeFunctionOptimalViewpoint.

- **Submòdul de plans:** la seva funció és mantenir i controlar tot el que fa referència a la visualització i el control dels plans de visualització; això inclou calcular l'*excess entropy* del punt de vista del pla. Està format per les classes OptimalViewpointPlane i OptimalViewpointPlaneHelper.

4.3.4 Mòdul de les classes compartides

Aquest mòdul conté les classes que no estan dissenyades específicament per a resoldre cap dels dos problemes principals, sinó que són útils per a coses més generals. Aquest mòdul té submòduls als paquets interface i tools. Podem veure el seu diagrama de classes a la Figura 5.26 (pàgina 88).

A interface:

- **Submòdul de definició de funcions de transferència:** està format per totes les classes que permeten definir una funció de transferència. Conté les classes HoverPoints, ShadeWidget i GradientEditor.

- **Submòdul de selecció de volums:** inclou les classes que permeten seleccionar un o més volums d'entre els que estan oberts. Conté les classes SelectVolumesDialogBase, SelectSingleVolumeDialog i SelectMultipleVolumesDialog.

A tools:

- **Submòdul d'interacció amb la visualització:** aquí hi ha classes que permeten interaccionar amb les finestres de visualització principals. Està format per les classes següents:

  - vtkInteractorStyleJoystickActorGgg

  - vtkInteractorStyleTrackballActorGgg

  - vtkInteractorStyleSwitchGgg

5 Disseny de l'aplicació

En aquest apartat explicarem el disseny de l'aplicació, explicant amb detall tots els mòduls i totes les classes que s'han creat per solucionar els problemes plantejats.

Primer de tot explicarem tots els detalls que cal saber sobre el disseny de la plataforma, ja que aquest condiciona el disseny de classes per implementar qualsevol algorisme que s'hi vulgui afegir, incloent els que hem implementat en aquest projecte. Un cop explicada l'estructura de la plataforma explicarem els detalls del disseny de les noves funcionalitats, incloent els mòduls, les classes i els mètodes, i les relacions entre els diferents mòduls i submòduls.

5.1 Disseny de la plataforma

L'estructura de classes de la plataforma és la que es pot veure en el diagrama de la Figura 5.1.

Cal dir que aquest és un diagrama simplificat, on només es mostren les classes bàsiques que pertanyen a la plataforma amb les connexions més importants entre elles. No es mostren les classes d'exemple, les que són mostres de com afegir noves funcionalitats a la plataforma. Tampoc es mostren les relacions amb classes externes, les que pertanyen a biblioteques que s'utilitzen, és a dir, classes de Qt, ITK, VTK, etc.

5.1.1 Estructura de la plataforma

De totes les classes de la plataforma, la més important és **QApplicationMainWindow**, que és la finestra principal de l'aplicació. Aquesta conté tot el que es pot esperar d'una finestra principal: menús, barres d'eines, barres d'estat, una àrea de treball, etc. A més de tots els elements bàsics d'una interfície d'usuari, conté les classes necessàries per obrir models, emmagatzemar-los, etc.

Els models de vòxels que s'obren queden encapsulats en la classe **Volume**. Aquesta classe pot proporcionar les dades del model en format ITK o VTK segons calgui, ja que internament fa les conversions necessàries.

Hi ha una QApplicationMainWindow oberta per cada model obert. Això no vol dir que aquella finestra sigui la "propietària" exclusiva del model obert. Es pot accedir a qualsevol model obert des d'una altra finestra mitjançant la classe **VolumeRepository**, que és un repositori comú on s'emmagatzemen tots els models oberts. Quan s'obre un nou model amb l'aplicació aquest s'afegeix al repositori, el qual retorna un identificador del model per poder-hi accedir. Cada finestra, doncs, guarda l'identificador del model que té obert. L'identificador està definit per la classe **Identifier**.

Només existeix una única instància del repositori per tota l'aplicació, independentment del nombre de finestres, ja que aquest està dissenyat seguin el patró **Singleton**. Per tant el repositori és accessible en qualsevol moment mitjançant un mètode de classe de VolumeRepository.

Malgrat tot, de moment el repositori només pot mantenir dos models alhora, per tant com a molt es podran obrir dues finestres simultàniament.

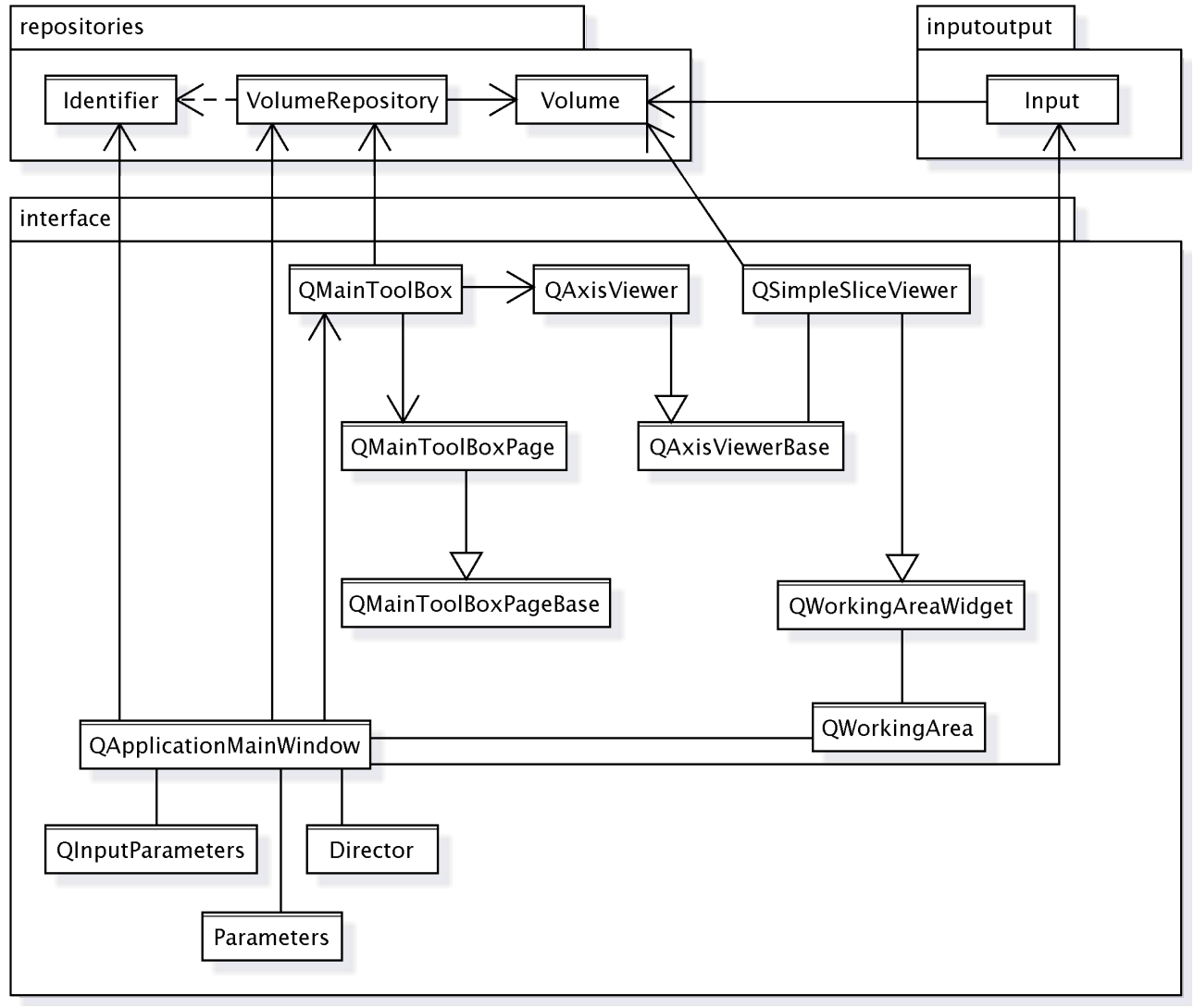


Figura 5.1: Diagrama de classes de la plataforma.

5.1.2 Paquets que constitueixen la plataforma

Com ja hem comentat abans, les classes de la plataforma estan repartides en diversos paquets, cadascun dels quals és un directori diferent al sistema de fitxers. Això ajuda a tenir el codi ordenat i classificat adequadament.

Els paquets són els següents:

- **main:** Aquí només hi ha el programa principal, que s'encarrega d'iniciar l'aplicació.
- **interface:** Aquí hi va tot el que sigui referent a la interfície d'usuari. Inclou tots els elements d'interfície, finestres, formularis, etc.
- **filters:** Destinat a agrupar filtres de caràcter general que s'aplicaran a les imatges.

- **colour:** Destinat a agrupar classes que tinguin a veure amb funcions de transferència i coses relacionades amb el color.
- **inputoutput:** Agrupa les classes que fan possible l'entrada/sortida de dades. Això va des de la lectura d'un model fins a l'escriptura d'imatges en un format determinat, com podria ser una imatge JPEG.
- **registration:** Agrupa els mètodes de registre que s'implementin.
- **segmentation:** Agrupa els mètodes de segmentació que s'implementin.
- **visualization:** Agrupa els mètode de visualització que s'implementin.
- **repositories:** Conté els repositoris de la plataforma. Actualment té el repositori de volums.
- **tools:** Conté utilitats a nivell general per tota la plataforma i les classes que van a cap dels altres paquets.

5.1.3 Finestra principal de l'aplicació

A la Figura 5.2 veiem l'aspecte que té la finestra principal de l'aplicació quan s'inicia. Com ja s'ha dit anteriorment, la finestra principal té els elements que cal esperar, com menús, barres d'eines, etc. A més a més d'aquests elements habituals cal destacar-ne dos en especial, la caixa d'eines i l'àrea de treball.

La **caixa d'eines**, implementada a la classe **QMainToolBox**, es troba a la part esquerra de la finestra principal i agrupa els formularis principals per escollir la tècnica de registre, segmentació o visualització que s'ha d'aplicar sobre el model i els seus respectius paràmetres. La caixa d'eines es divideix en pàgines (**QMainToolBoxPage**) que agrupen els mètodes de registre, segmentació i visualització, a més a més d'una pàgina amb un visor encastat de les vistes axial, coronal i sagital del model.

Per fer els formularis d'introducció de paràmetres pels nous mètodes que puguin afegir-se a la caixa d'eines, aquests formularis han d'heretar de la classe **QInputParameters**. D'aquesta manera heretaran un conjunt de *signals* i *slots* necessaris per fer la comunicació amb altres parts de l'aplicació.

L'**àrea de treball**, implementada a la classe **QWorkingArea**, és la part més gran de la finestra principal, la zona grisa central. És el lloc indicat per a posar les noves finestres que s'obrin, tant de visualització com de resultats numèrics. Aquesta classe s'encarrega del posicionament de les finestres entre altres coses.

5.1.4 Integració de nous mètodes a la plataforma

Un cop vists els components més importants de la plataforma, ja podem explicar quina és l'estructura de classes que ha de seguir un nou mètode que vulguem integrar a la plataforma. Concretament, les classes que cal crear per cada mètode nou que es vulgui afegir són:

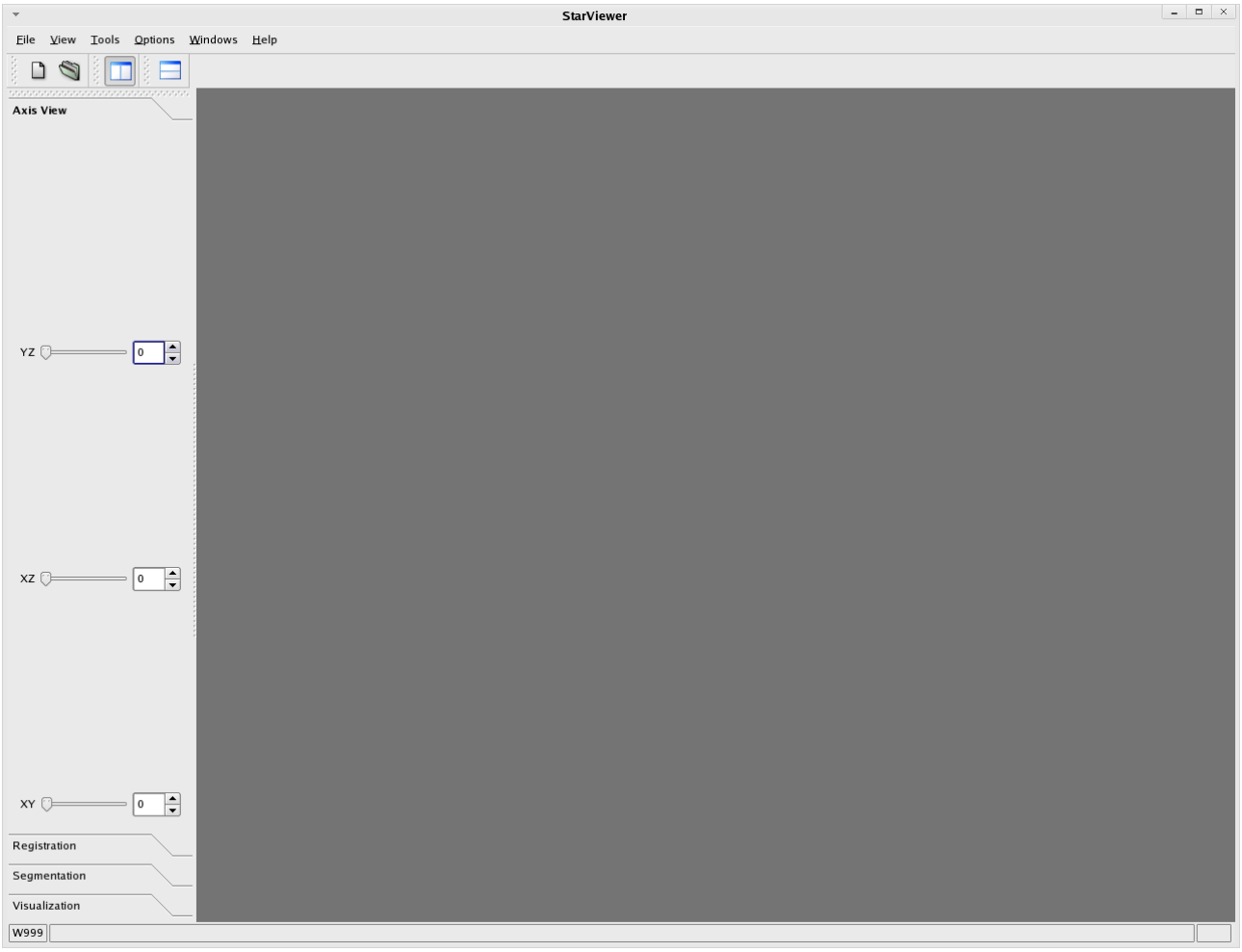


Figura 5.2: Finestra principal de la plataforma acabada d'obrir.

- **Una o més classes d'implementació del mètode de registre, segmentació o visualització.** La classe principal d'aquestes ha de portar un nom descriptiu del mètode, per exemple **Asdf**.

- **Una classe d'encapsulament dels pàrametres.** Aquesta classe ha d'heretar de la classe **Parameters** i s'anomenarà **AsdfParameters**. Aquesta classe ha de guardar tots els paràmetres d'entrada del mètode i ha de tenir una enumeració anomenada **AsdfParametersNames** que tingui com a valors el nom de cada paràmetre, per exemple ParameterA, ParameterS, ParameterD, ParameterF. També hi ha d'haver un mètode *set* i un *get* per cada paràmetre. Els mètodes *set* han d'emetre el *signal* **changed(int)** utilitzant com a argument el nom del paràmetre modificat, com a l'enumeració.

- **Un o més formularis d'introducció de paràmetres.** Pels formularis de més alt nivell, que s'hauran d'afegir a la caixa d'eines, caldrà:

  - Crear un formulari base que es limiti a definir l'aparença del formulari, amb els seus *widgets* i *layouts*. El més recomanable és fer servir el Qt Designer per crear un fitxer .ui que després serà convertit en una classe per l'eina `uic` de les Qt. Aquest formulari base s'ha d'anomenar **AsdfInputParametersFormBase**.

  - Crear una classe d'implementació del formulari que hereti de la que crea l'`uic` i de **QInputParameters**. Aquesta nova classe ha d'implementar totes les funcionalitats que calguin relacionades amb el formulari base: connexions de *signals i slots*, mètodes de validació de dades, etc. La classe ha d'implementar també dos mètodes abstractes de QInputParameters: **readParameter(int)** i **writeAllParameters()**. El primer ha de llegir un paràmetre determinat cridant el mètode *get* corresponent de la classe AsdfParameters. El segon escriu tots els paràmetres cridant els mètodes *set* corresponents de la classe AsdfParameters. També hi ha d'haver un mètode **setParameters(AsdfParameters*)** per assignar-li un punter a la classe de paràmetres que ha de guardar per fer servir. Aquesta classe s'ha de dir **AsdfInputParametersForm**.

- **Una classe d'enllaç entre la interfície i el mètode implementat.** Aquesta classe heretarà de **Director** i es dirà **AsdfDirector**. Hereta de Director un punter a QApplicationMainWindow i ha de guardar-ne un a AsdfParameters i un a Asdf. Aquesta classe també ha de tenir un mètode **setParameters(AsdfParameters*)** per assignar-li l'objecte de paràmetres que ha d'utilitzar. Però el mètode més important que ha d'implementar és l'**execute()**, heretat de Director, que és un *slot*. Aquest mètode és el que crea l'objecte del mètode, li dóna els paràmetres, el fa executar i en mostra els resultats, creant les finestres necessàries. Les finestres s'afegeixen mitjançant el mètode **addWorkingAraWidget(...)** de QApplicationMainWindow.

- **Finestres de resultats.** S'han de mostrar a l'àrea de treball, i perquè això sigui possible han d'heretar de **QWorkingAreaWidget**.

Un cop creades totes les classes, cal enllaçar-les amb la plataforma. Per fer-ho cal modificar **QApplicationMainWindow**. Caldrà afegir-hi punters a AsdfDirector, AsdfParameters i AsdfInputParametersForm. També cal afegir-hi un punter a **Q3Action**, que guardarà el l'acció que cridarà el mètode execute() d'AsdfDirector. Al **constructor** s'hauran d'instanciar tots aquests objectes, passar l'objecte de paràmetres als altres dos objectes i fer algunes connexions de *signals i slots*:

- Connectar el *signal* activated() de l'acció amb l'*slot* execute() d'AsdfDirector.

- Connectar el *signal* changed(int) d'AsdfParameters amb l'*slot* readParameter(int) d'AsdfInputParametersForm.

Per acabar, al mètode **createMainToolBox()** s'haurà d'afegir el formulari a la caixa d'eines mitjançant el mètode **addWidget(...)** de **QMainToolBox**.

Amb tot això el nou mètode quedarà completament integrat a la plataforma. El que falta dir és que l'acció s'activarà quan l'usuari premi el botó **"Apply"** de **QMainToolBoxPageBase**. El botó es troba a la part de baix i a la dreta de cada pàgina de la caixa d'eines.

Podem veure un esquema diagrama de classes d'aquest exemple a la Figura 5.3, on Asdf és un mètode de segmentació. Les classes blanques són les que té la plataforma bàsica, les grises són de Qt i les grogues són les que s'han creat per aquest mètode.

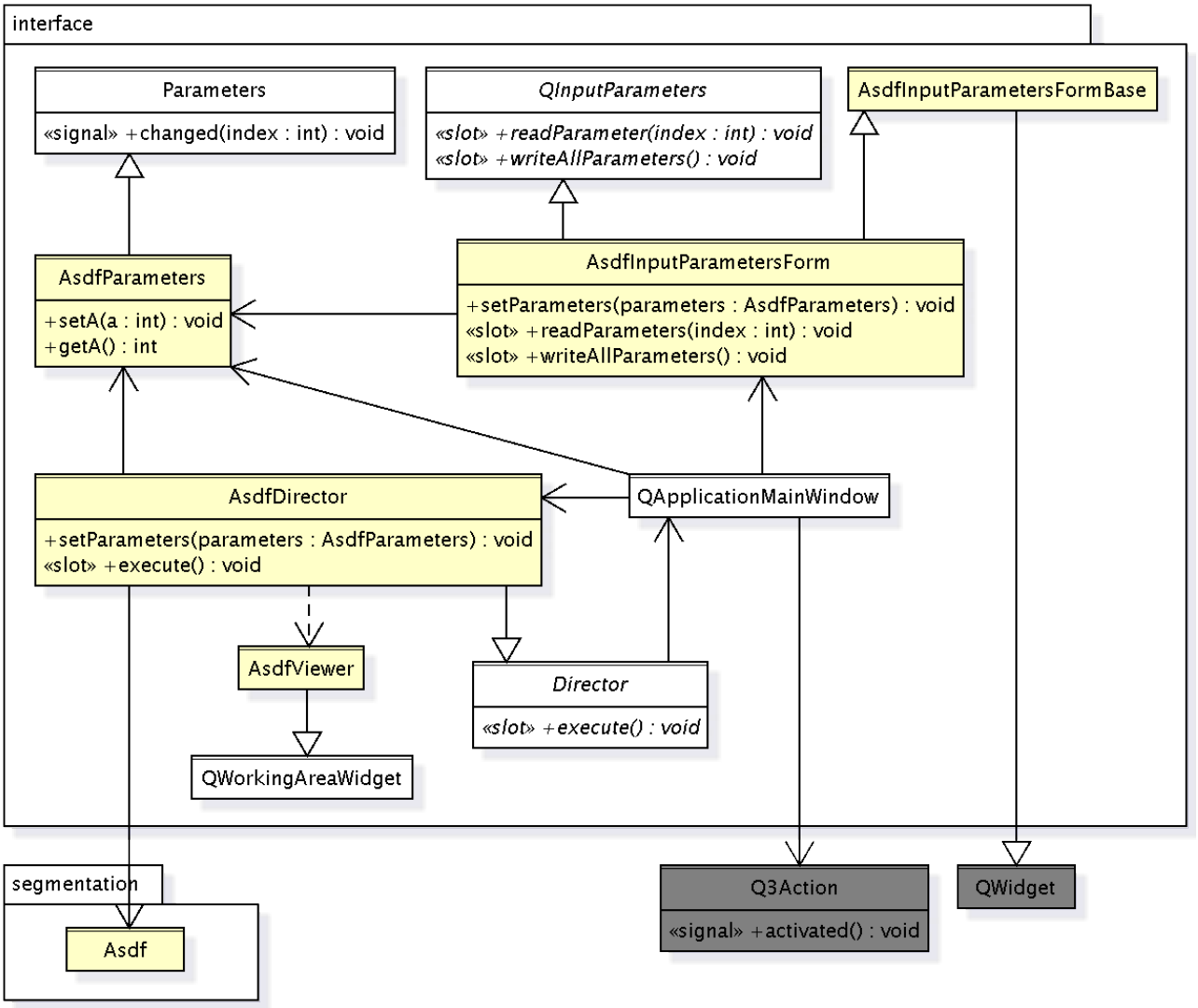


Figura 5.3: Diagrama de classes d'un mètode de segmentació qualsevol anomenat Asdf.

Aquesta manera d'integrar nous mètodes a la plataforma justifica el disseny dels dos mòduls principals que hem creat, el dels Miralls Màgics i el de la selecció del punt de vista òptim. Tots dos tenen una estructura semblant, amb una divisió en submòduls semblant.

Els submòduls de la interfície corresponen als diferents tipus de classes que cal crear per la interfície: la classe de paràmetres, el/s formulari/s d'introducció de paràmetres, la classe d'enllaç entre la interfície i el mètode, i la finestra de visualització de resultats.

D'altra banda, també s'han dividit en submòduls les classes que implementen el mètode, però en aquest cas no té res a veure amb l'estructura de la plataforma, sinó que és una decisió de disseny, per agrupar les classes que estan relacionades més estretament.

Per acabar, el mòdul de les classes compartides no implementa cap mètode nou, per tant té una estructura completament diferent.

Un cop justificat el disseny de l'aplicació, ja podem explicar en detall cadascun dels mòduls que la composen, llevat del mòdul de la plataforma que ja ha quedat explicat.

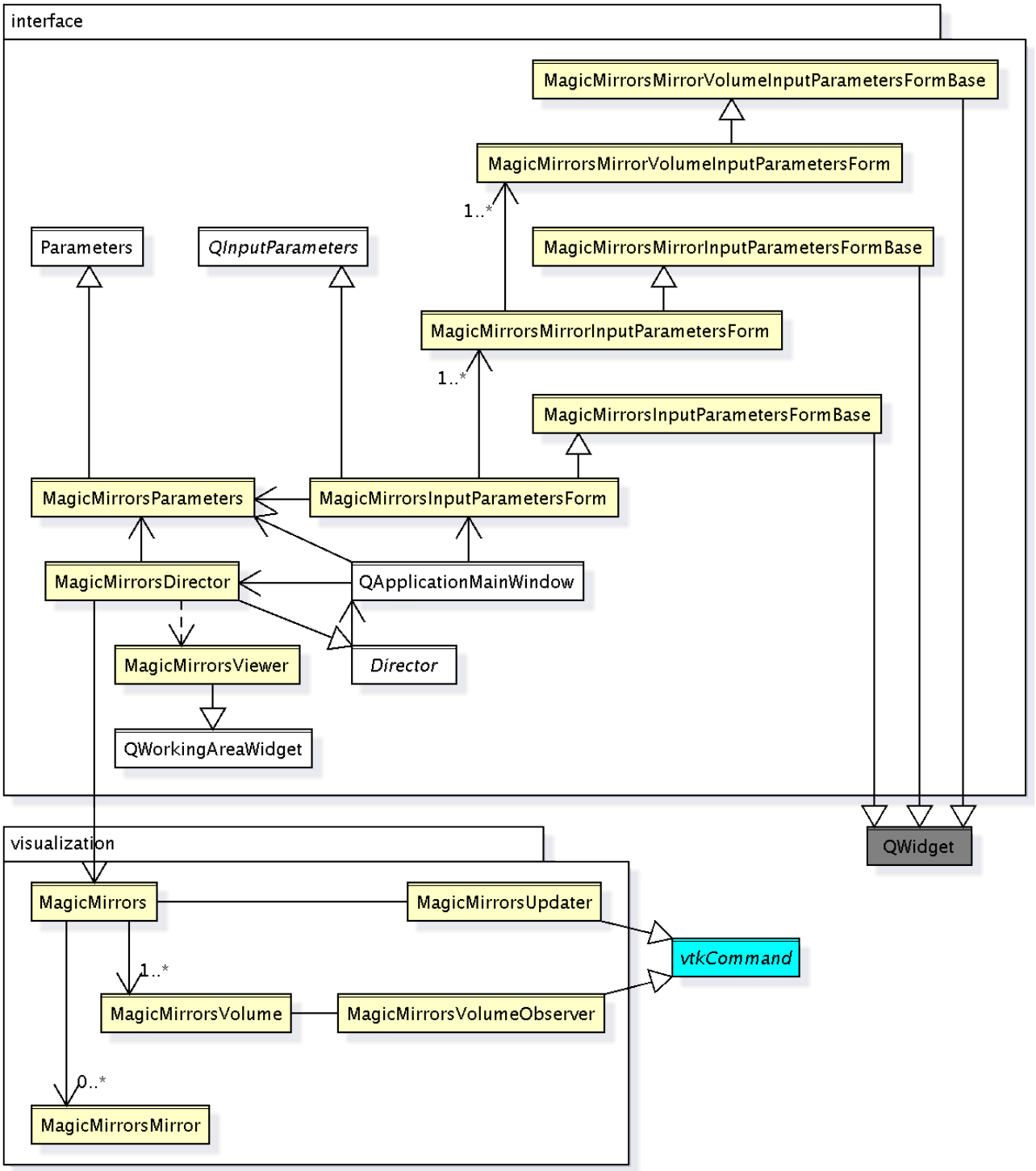


Figura 5.4: Diagrama de classes del mòdul dels Miralls Màgics.

5.2 Disseny del mòdul dels Miralls Màgics

Com hem dit abans, aquest mòdul conté totes les classes que estan dissenyades específicament per implementar els Miralls Màgics, resolent el problema de la visualització de models fusionats. El seu diagrama de classes general és el que es pot veure a la Figura 5.4. L'esquema de colors és el mateix, afegint que les classes en cian clar són de VTK.

Podem veure que és un diagrama semblant al de l'exemple d'abans (Figura 5.3). La diferència principal és que hi ha moltes més classes per construir el formulari d'introducció de paràmetres i també més classes per implementar el mètode.

Veiem ara en detall cada submòdul. Però abans cal comentar que tot i les limitacions de la plataforma de poder obrir només dos volums, a l'hora de dissenyar aquest mòdul hem fet les totes les classes pensant en que se'n puguin obrir tants com es vulgui, perquè no calgui modificar les classes si en un futur la plataforma permet obrir més volums.

5.2.1 Disseny del submòdul d'introducció de paràmetres

El diagrama de classes específic d'aquest submòdul és el de la Figura 5.5. Es manté l'esquema de colors però ara també hi ha classes d'STL, que són en verd pastel.

Com ja hem dit, les classes d'aquest submòdul permeten introduir tots els paràmetres d'entrada dels Miralls Màgics. Són tres parells de classes, on cada parell ocupa un nivell diferent de jerarquia.

MagicMirrorsMirrorVolumeInputParametersFormBase i MagicMirrorsMirrorVolumeInputParametersForm

Aquestes són les classes del nivell més baix i permeten introduir els paràmetres que afecten a un volum determinat en un mirall determinat. Tots aquests paràmetres són de visualització, i són els següents:

- **Visibilitat:** és un booleà que indica si aquest volum es veurà en aquest mirall. Quan estem treballant amb models fusionats això ens permet definir quines propietats es veuen a cada volum: només caldrà desactivar aquesta opció dels volums corresponents a les propietats que no vulguem que es vegin.

- **Ombrejat:** és un booleà que indica si aquest volum es pintarà amb ombrejat en aquest mirall. L'ombrejat fa que el volum es vegi millor en alguns casos, però alenteix el procés de visualització. Podem veure la diferència entre els dos tipus de visualització a la Figura 5.6.

- **Funció de transferència:** és un objecte de tipus MagicMirrors::TransferFunction, que és un sinònim per a QGradientStops. És la funció de transferència que s'aplicarà a aquest volum en aquest mirall. Aquesta funció de transferència està definida en l'espai [0,1] × RGBA.

La classe base és la que defineix el formulari: els *widgets* que s'utilitzen i com es distribueixen. Hi ha un *widget* especialment important que és GradientEditor, el qual permet definir o modificar una

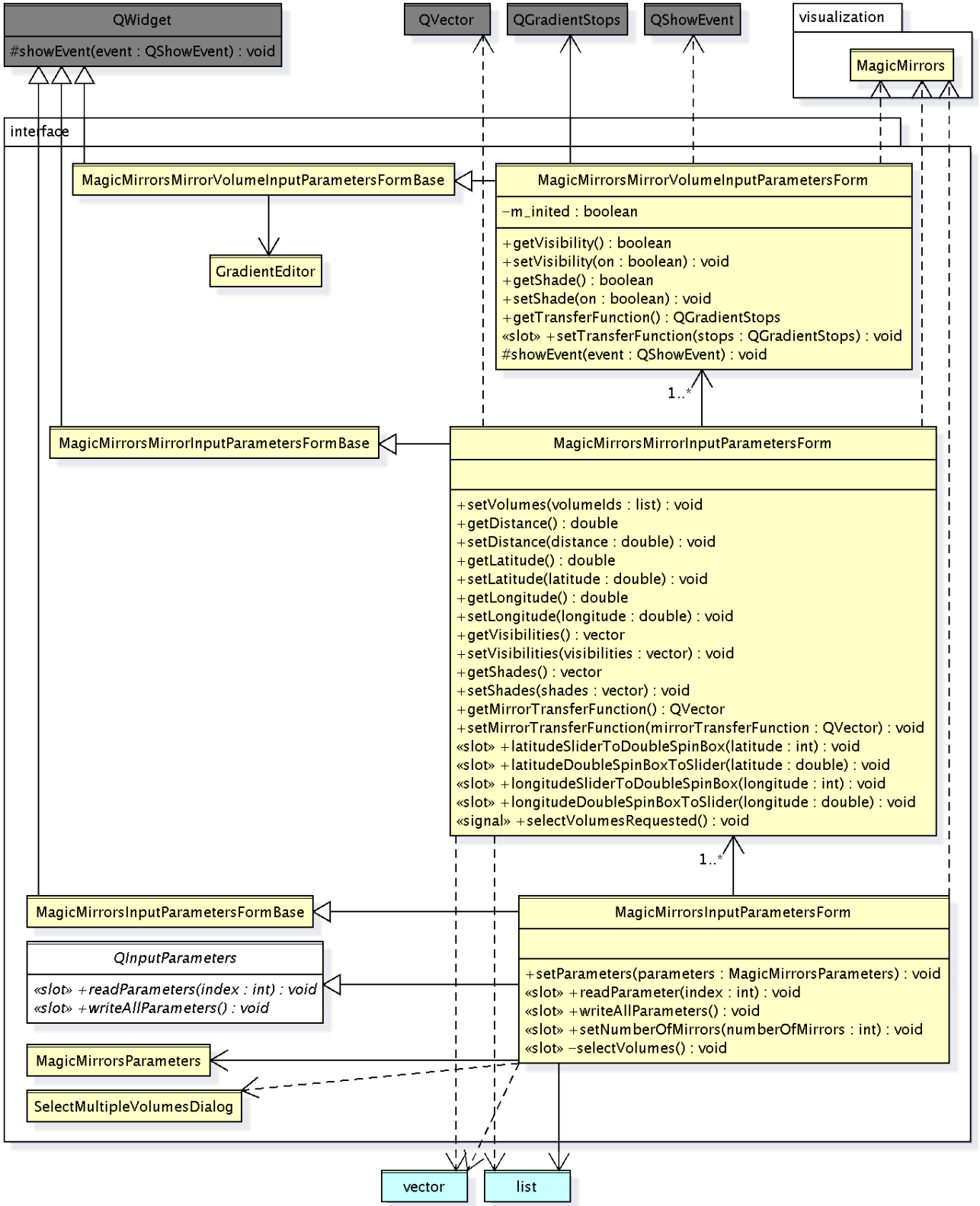


Figura 5.5: Diagrama de classes del submòdul d'introducció de paràmetres dels Miralls Màgics.

funció de transferència d'una manera molt senzilla i intuïtiva. En veurem els detalls quan expliquem el disseny del mòdul de les classes compartides. El que interessa saber ara és que quan es modifica la

funció de transferència emet un *signal* amb un objecte QGradientStops com a paràmetre. Aquest *signal* es connecta a l'*slot* setTransferFunction(const QGradientStops&) i d'aquesta manera es comuniquen.

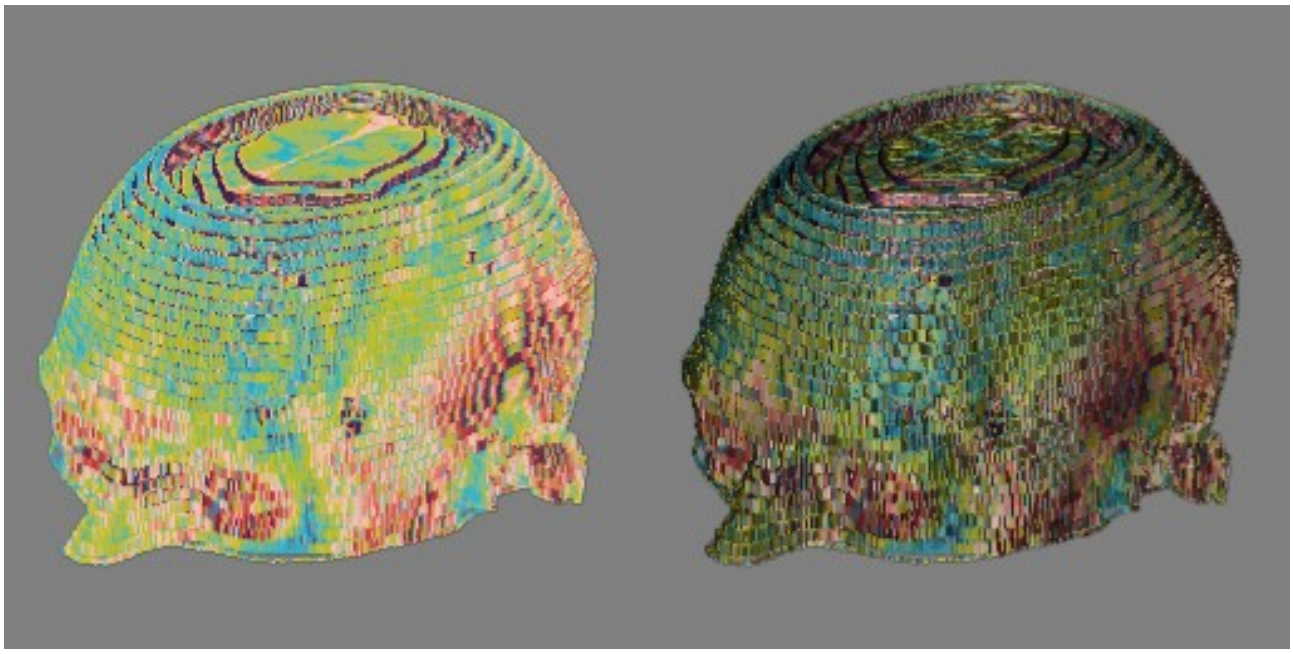


Figura 5.6: Demostració del que fa l'ombrejat. A l'esquerra, sense ombrejat. A la dreta, amb ombrejat.

La classe filla fa la implementació del formulari i té mètodes *set* i *get* per cada paràmetre. També reimplementa el mètode showEvent(QShowEvent*) de QWidget per fer la inicialització correcta del GradientEditor: el que fa és cridar el mètode de QWidget i la primera vegada que es mostra el formulari (controlat per l'atribut m_inited) inicialitza el GradientEditor amb la funció de transferència per defecte[3].

MagicMirrorsMirrorInputParametersFormBase i MagicMirrorsMirrorInputParametersForm

Aquestes són les classes del nivell del mig i permeten introduir els paràmetres que afecten a un mirall determinat. Aquests paràmetres inclouen els que serveixen per definir la posició del mirall i són els següents:

- **Distància des del centre:** és un real que indica a quina distància del centre de l'espai[4] es situarà el mirall.

- **Latitud:** és un real que indica la latitud on es situarà el mirall, en graus. Aquesta latitud és una analogia de la latitud d'un punt de la Terra, i pot anar de -90 (pol sud) a 90 (pol nord).

3 El GradientEditor no s'inicialitza bé si no s'ha mostrat almenys una vegada.
4 El centre de l'espai és el punt (0,0,0).

- **Longitud:** és un real que indica la longitud on es situarà el mirall, en graus. Aquesta longitud és una analogia de la longitud d'un punt de la Terra, i pot anar de -179.9 a 180.

La classe base és la que defineix el formulari: els *widgets* que s'utilitzen i com es distribueixen. La distància s'introdueix mitjançant un *double spin box*, i tant la latitud com la longitud es poden introduir amb un *slider* o bé un *double spin box*. A banda d'aquests, hi ha un botó "Select..." que permet seleccionar els volums que s'han de visualitzar, un *combo box* per triar el volum concret que es vol configurar i un formulari del nivell inferior per cada volum, dels quals només se'n veu un cada vegada, el que s'ha seleccionat amb el *combo box*.

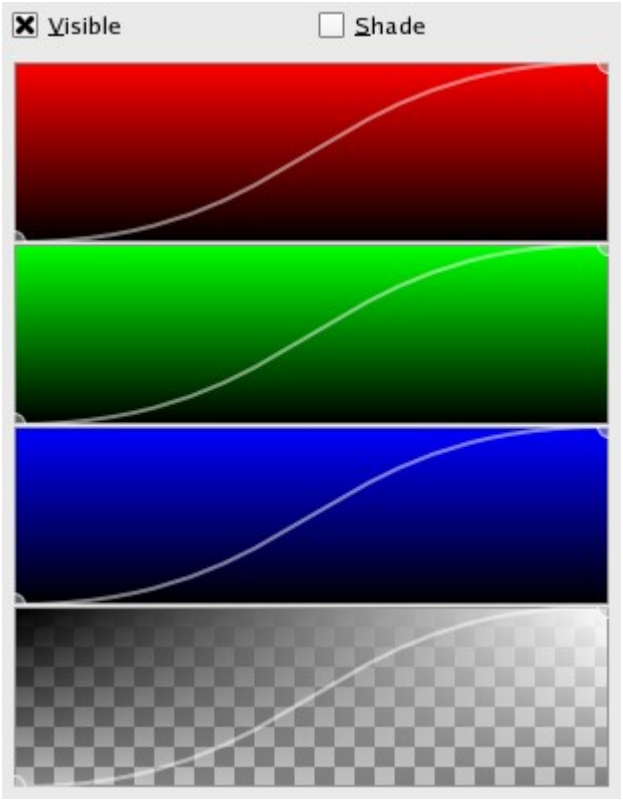


Figura 5.7: MagicMirrorsMirrorVolumeInputParametersForm.

La classe filla fa la implementació del formulari i té mètodes *set* i *get* per cadascun dels paràmetres d'aquest nivell. També té *sets* i *gets* pels paràmetres del nivell inferior, però en aquest cas treballa amb vectors perquè hi ha d'haver un valor per cada volum.

Aquesta classe filla també conté un mètode anomenat setVolumeIds(std::list<int>*) que serveix per assignar-li la llista dels identificadors dels volums seleccionats per l'usuari. Aquest mètode serà cridat per MagicMirrorsInputParametersForm quan l'usuari acabi la selecció dels volums. Per demanar la selecció dels volums s'emet el *signal* selectVolumesRequested() quan l'usuari prem el botó "Select...". Un cop s'hagi assignat la llista de volums el botó s'inhabilita. També s'omple el *combo box* que permet triar el volum que es vol configurar i es creen els formularis del nivell inferior per configurar cada volum.

Per acabar, també hi ha quatre *slots* per sincronitzar els dos *widgets* d'entrada de la latitud i els dos de la longitud. Això és necessari perquè uns treballen amb enters i els altres amb reals.

MagicMirrorsInputParametersFormBase i MagicMirrorsInputParametersForm

Aquestes són les classes del nivell més alt i permeten introduir els paràmetres que afecten al mètode de visualització en si mateix. Els paràmetres són els dos següents:

- **Volums a visualitzar:** és una llista amb els identificadors (enters) dels volums que es visualitzaran.

- **Nombre de miralls:** és un enter que indica el nombre de miralls que hi ha d'haver. El mínim són 0 i el màxim 20.

La classe base és la que defineix el formulari: els *widgets* que s'utilitzen i com es distribueixen. Hi ha un *spin box* per introduir el nombre de miralls i un conjunt de pestanyes per accedir a cadascun dels formularis del nivell inferior, dels quals n'hi ha un per configurar el model central i un per cada mirall.

La classe filla fa la implementació del formulari. S'encarrega de crear els formularis del nivell inferior, dels quals el primer correspon al volum central i per tant té els controls de la posició inhabilitats.

Una altra tasca que fa és mostrar el diàleg de selecció de volums (SelectMultipleVolumesDialog) i assignar els volums seleccionats als formularis del nivell inferior. Tot això ho fa al mètode selectVolumes(), que és un *slot* que es crida en rebre el *signal* de petició d'algun dels MagicMirrorsMirrorInputParametersForms.

Finalment hi ha els dos *slots* heretats de QInputParameters, amb els quals es comunica amb MagicMirrorsParameters per llegir o escriure qualsevol paràmetre. En aquest cas són els paràmetres d'aquest nivell (volums, nombre de miralls), que són directes, els del nivell inferior (distàncies, latituds, longituds), que són amb vectors, i els del nivell inferior de l'inferior (visibilitats, ombrejats, funcions de transferència), que són amb vectors de vectors.

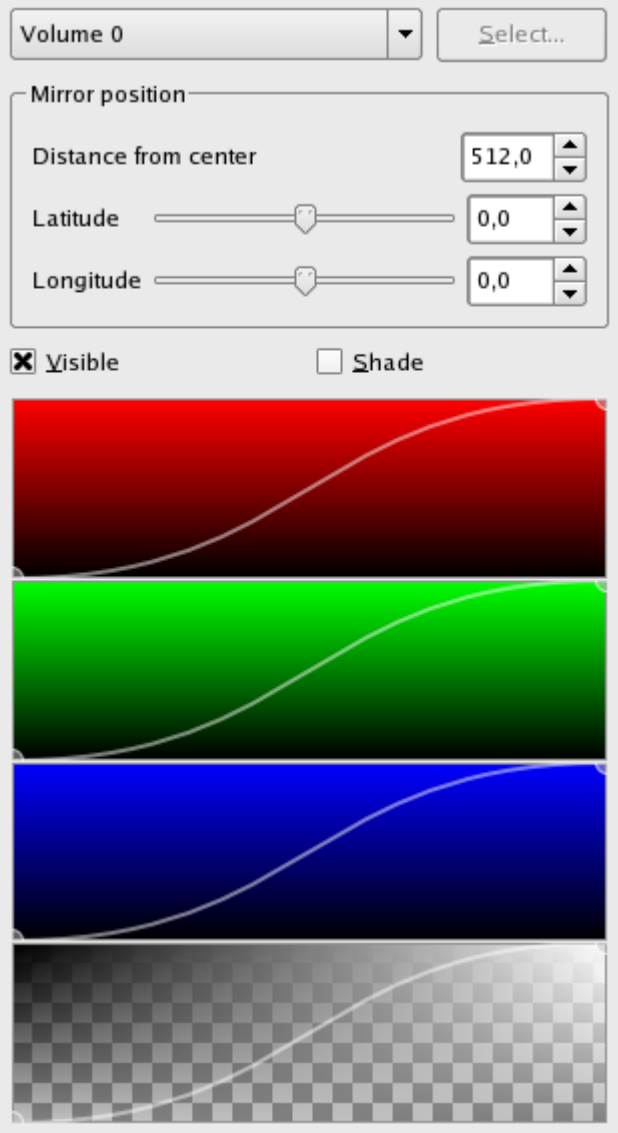


Figura 5.8: MagicMirrorsMirrorInputParametersForm.

5.2.2 Disseny del submòdul d'encapsulament de paràmetres

El diagrama de classes específic d'aquest submòdul és el de la Figura 5.10.

Com ja hem dit, la funció d'aquest submòdul és guardar tots els paràmetres d'entrada dels Miralls Màgics per mantenir-los tots en un sol lloc. L'única classe d'aquest mòdul és **MagicMirrorsParameters**, que és una subclasse de Parameters.

Fem un breu recordatori dels paràmetres d'aquest mètode:

- Identificadors dels volums seleccionats
- Nombre de miralls
- Distància des del centre de cada mirall
- Latitud de cada mirall
- Longitud de cada mirall
- Visibilitat de cada volum a cada mirall
- Ombrejat de cada volum a cada mirall
- Funció de transferència de cada volum a cada mirall

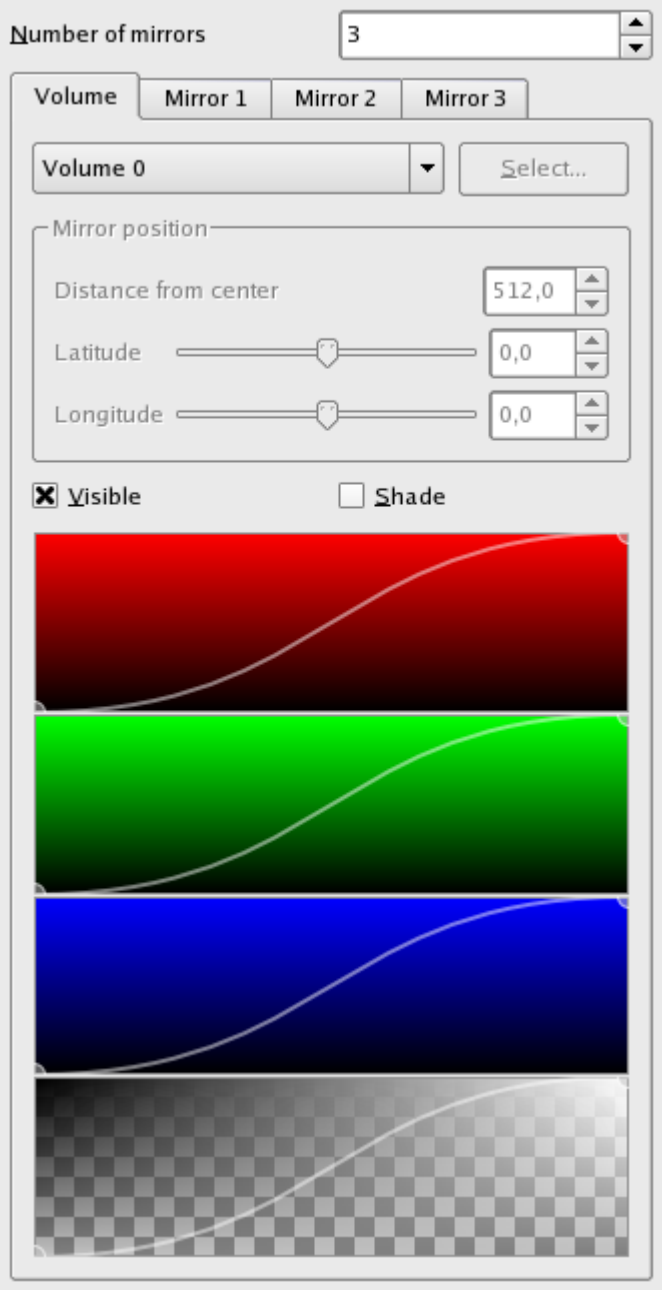


Figura 5.9: MagicMirrorsInputParametersForm.

MagicMirrorsParameters defineix una enumeració amb els noms dels paràmetres, MagicMirrorsParametersNames, amb els elements següents: VolumeIds, NumberOfMirrors, Distances, Latitudes, Longitudes, Visibilities, Shades, TransferFunctions.

Guarda els paràmetres en atributs. Els del mètode els guarda directament, els dels miralls els guarda en vectors i els dels volums els guarda en vectors de vectors. Per cada paràmetre hi ha els mètodes *set* i *get* corresponents. Els *sets* emeten el *signal* changed(int) heretat de Parameters, amb el nom del paràmetre (agafat de MagicMirrorsParametersNames) com a argument. Aquest *signal* serà recollit per MagicMirrorsInputParametersForm i li dirà que ha de llegir el paràmetre indicat per l'argument del *signal*.

Aquest submòdul no té res més a destacar perquè només es tracta del magatzem de paràmetres del mètode.

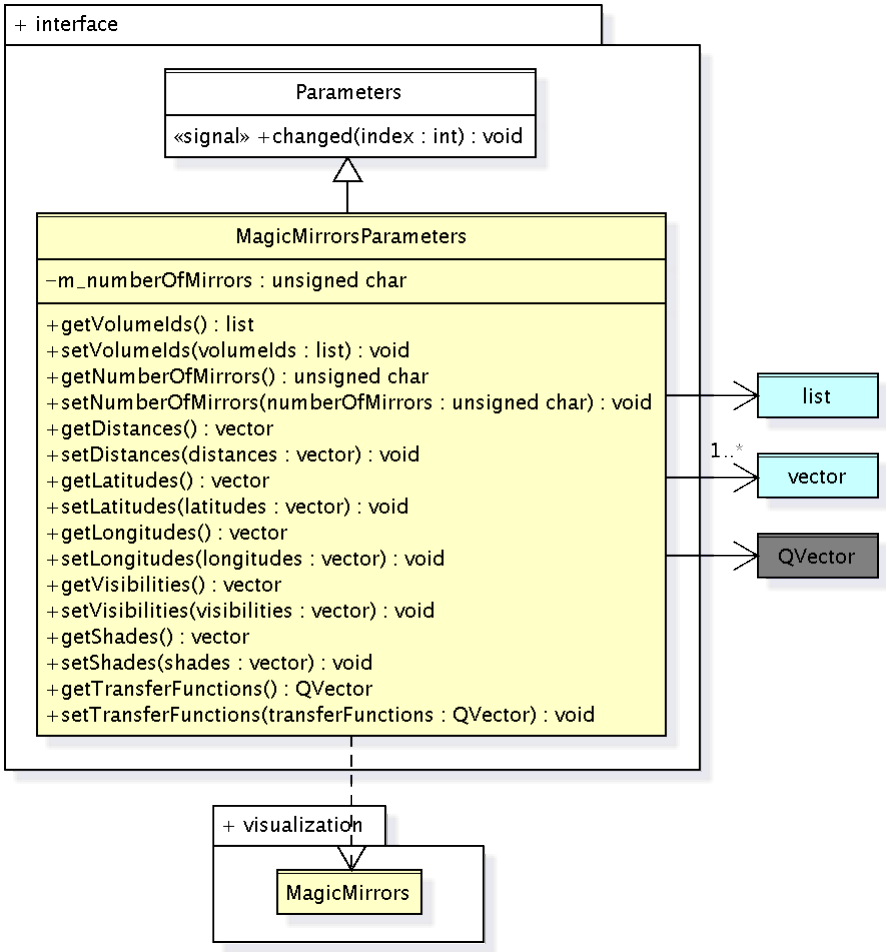


Figura 5.10: Diagrama de classes del submòdul d'encapsulament de paràmetres dels Miralls Màgics.

5.2.3 Disseny del submòdul de la finestra de visualització principal

El diagrama de classes específic d'aquest submòdul és el de la Figura 5.11.

Com ja hem dit, aquest submòdul consisteix únicament en la finestra de visualització principal del mètode de visualització dels Miralls Màgics, on es veu el model amb els miralls al voltant. El nom de la classe que implementa aquesta finestra és **MagicMirrorsViewer**.

La finestra ha d'aparèixer a l'àrea de treball de l'StarViewer, i per aquesta raó és una subclasse de la classe QWorkingAreaWidget.

La classe és mol senzilla, ja que només es tracta d'una finestra. Té com a únics atributs un QVTKWidget, on es veurà la visualització, i un vtkRenderer, que és el que s'encarrega de dibuixar la visualització.

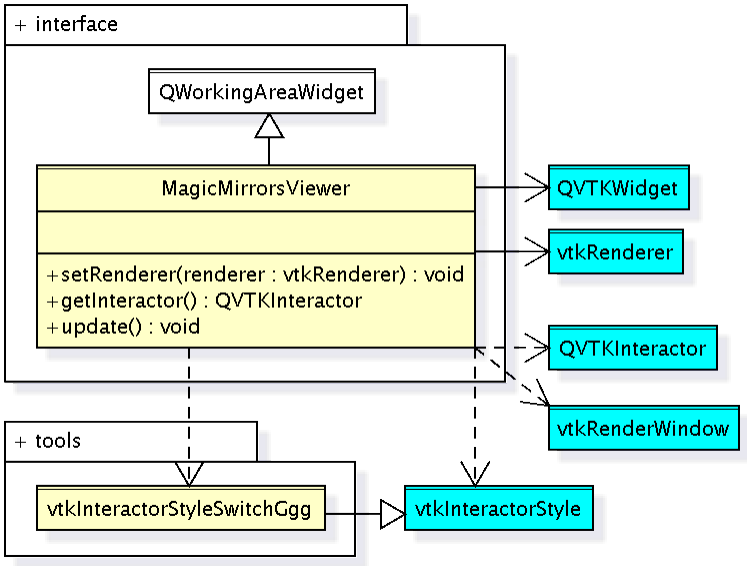


Figura 5.11: Diagrama de classes del submòdul de la finestra de visualització principal dels Miralls Màgics.

Té un mètode per assignar-li el *renderer* que ha de fer servir, un per obtenir l'*interactor* (l'objecte que permet interaccionar directament amb la visualització) i un per forçar l'actualització de la visualització.

Podem destacar com a particularitat que fa servir la classe vtkInteractorStyleSwitchGgg, del paquet tools, en comptes del vtkInteractorStyleSwitch de VTK. El motiu és que la versió GGG, que és una modificació de l'original, té un temps de resposta més petit quan es vol interaccionar amb el model, però això ho explicarem amb detall quan expliquem el disseny del mòdul de les classes compartides.

A banda d'això, no hi ha res més a destacar en aquest submòdul.

5.2.4 Disseny del submòdul d'enllaç de la interfície amb les classes de control

El diagrama de classes específic d'aquest submòdul és el de la Figura 5.12.

Com ja hem dit, aquest submòdul s'encarrega de relacionar les classes que formen part de la interfície amb les classes de control, que són les que fan la visualització i es troben al paquet visualization. L'única classe d'aquest submòdul és **MagicMirrorsDirector**.

Aquesta classe és filla de Director i d'aquest hereta l'*slot* execute(), el qual ha d'implementar perquè executi el mètode. També disposa d'un mètode perquè li assignin l'objecte de paràmetres.

Per poder realitzar la seva feina de coordinació, aquesta classe manté punters al mètode (MagicMirrors), als paràmetres (MagicMirrorsParameters) i a la finestra de visualització principal (MagicMirrorsViewer), així com un punter a la finestra principal de l'aplicació (QApplicationMainWindow) heretat de Director.

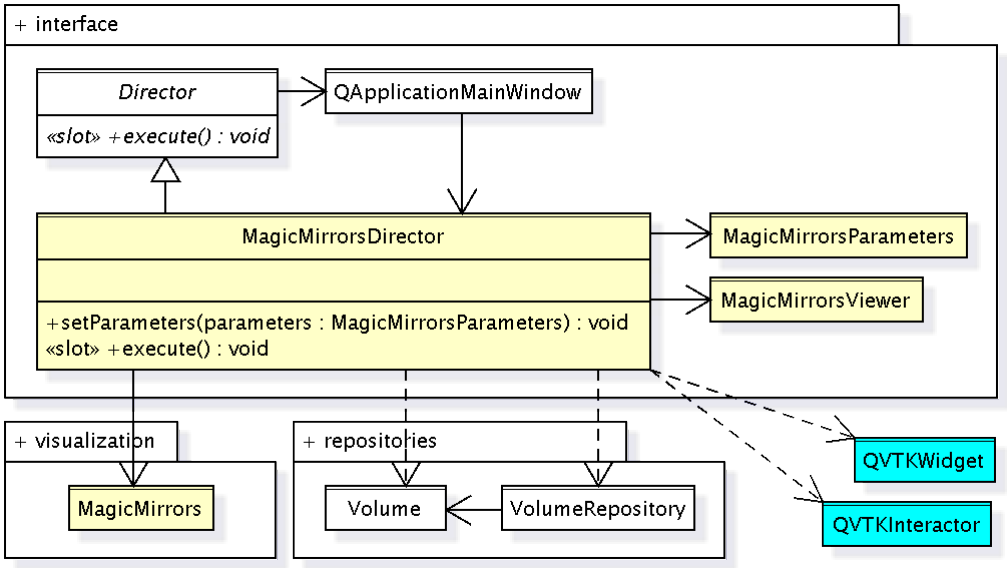


Figura 5.12: Diagrama de classes del submòdul d'enllaç de la interfície amb les classes de control dels Miralls Màgics.

La primera vegada que es crida l'execute(), crea el mètode, i també crea el visor i l'afegeix a l'àrea de treball. També afegeix els volums seleccionats al mètode, agafant-los del repositori mitjançant l'identificador.

La resta de tasques les fa cada vegada, i consisteixen en passar cada paràmetre al mètode, cridar el mètode per actualitzar els miralls i cridar el mètode del visor per actualitzar la visualització.

Això és tot el que fa aquest submòdul.

5.2.5 Disseny del submòdul del control principal de la visualització

El diagrama de classes específic d'aquest submòdul és el de la Figura 5.13.

Com ja hem dit, aquest submòdul està format per les classes que controlen la visualització principal dels Miralls Màgics. Aquestes classes són **MagicMirrors** i **MagicMirrorsUpdater**.

MagicMirrors és la classe principal, la que fa realment la feina. MagicMirrorsUpdater és un ajudant de MagicMirrors, que crida alguns dels seus mètodes d'actualització quan cal.

MagicMirrors guarda un conjunt de volums (MagicMirrorsVolume) i s'encarrega de gestionar-los. Manté tots els volums sincronitzats, en el sentit que tots estan a la mateixa posició i amb la mateixa rotació. D'aquesta manera es comporten com un sol volum de cara a l'usuari, de manera que aquest els percep com un sol model fusionat. D'altra banda, cal que els volums s'hagin registrat prèviament, abans d'iniciar aquest mètode, ja que els Miralls Màgics només són un mètode de visualització i no hi ha cap manera de detectar si els volums estan registrats o no.

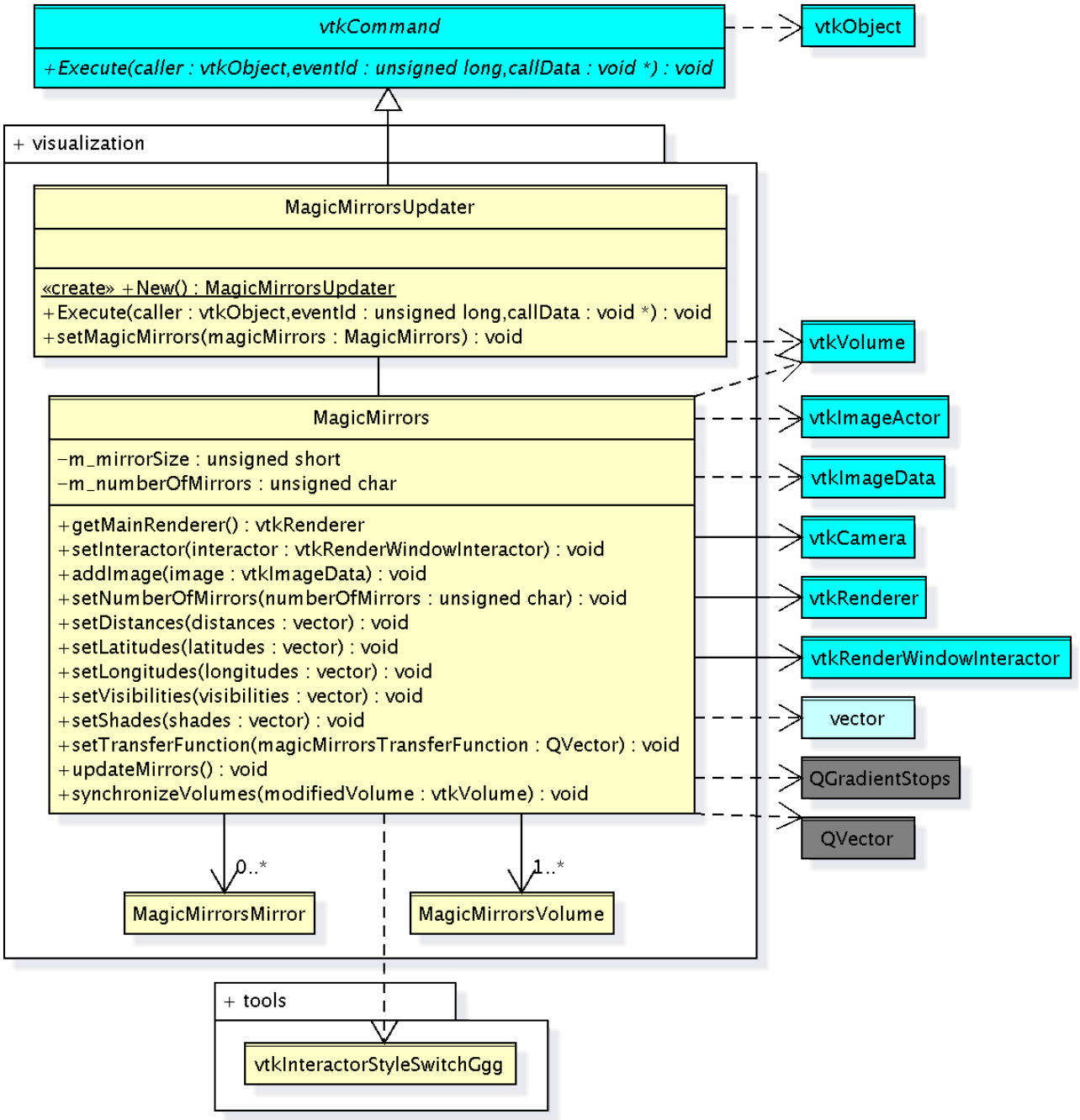


Figura 5.13: Diagrama de classes del submòdul del control principal de la visualització dels Miralls Màgics.

La classe també guarda un conjunt de miralls (MagicMirrorsMirror). Passa els paràmetres que li corresponen a cadascun i els fa actualitzar quan cal, a més a més de fer que es visualitzin a la finestra principal.

És MagicMirrors qui s'encarrega de crear el *renderer* principal (el de la finestra de visualització principal) i el retorna amb el mètode getMainRenderer(). Per fer que es visualitzi un objecte, sigui un volum o un mirall, el que fa és afegir-lo al *renderer*, i per aconseguir que no es visualitzi el treu del

renderer. Amb això cal anar amb compte perquè s'han d'afegir primer els miralls i després els volums, o sinó els miralls es dibuixaran sempre per davant dels volums.

D'altra banda, la classe MagicMirrors necessita tenir l'*interactor* per afegir-hi un observador, el MagicMirrorsUpdater. Es tracta que el MagicMirrorsUpdater vigili l'*interactor*, tot aplicant el patró de disseny Observer, i respongui a uns esdeveniments determinats. En aquest cas és l'esdeveniment de fi d'interacció. Quan l'*interactor* generi aquest esdeveniment es cridarà el mètode Execute(...) de MagicMirrorsUpdater, i aquest cridarà el mètode updateMirrors() de MagicMirrors. Amb tot això quan l'usuari acabi una interacció amb el model s'actualitzaran els miralls automàticament. El patró Observer està implementat a la classe vtkCommand, que és la superclasse de MagicMirrorsUpdater.

El penúltim aspecte a destacar de MagicMirrors és que redueix els valors de propietat dels models que se li afegeixen a un rang entre 0 i 255. Això facilita que la resta de classes puguin treballar sempre amb un rang de valors conegut i és suficient per veure bé la imatge. També cal dir que s'agafa com a mida de mirall la diagonal del primer model afegit.

L'última funcionalitat d'aquesta classe és mantenir tots els volums d'un model fusionat correctament sincronitzats (alineats). Això ho fa amb el mètode synchronizeVolumes(vtkVolume*), que és cridat

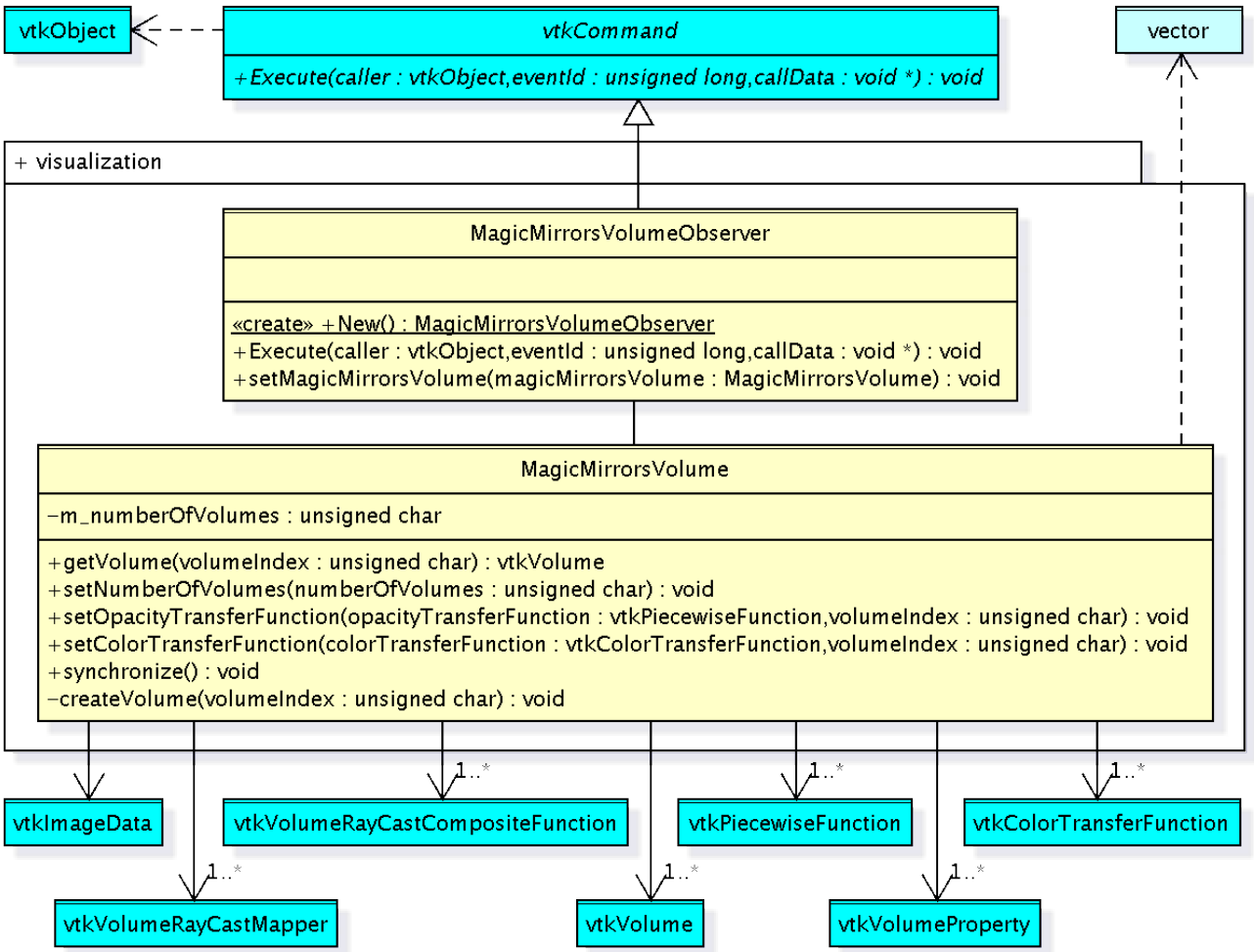


Figura 5.14: Diagrama de classes del submòdul de volums dels Miralls Màgics.

també per MagicMirrorsUpdater. En aquest cas espera un esdeveniment de modificació d'un dels volums. Per tant, quan l'usuari mou o gira un dels volums del model fusionat, MagicMirrorsUpdater crida el mètode abans esmentat passant com a paràmetre el volum modificat i MagicMirrors s'encarrega d'alinear tots els volums d'acord amb el modificat. Tot això passa mentre l'usuari està manipulant el model, és a dir, que la sincronització és contínua.

Per acabar amb aquest mòdul direm que MagicMirrors té mètodes per assignar cada paràmetre, en els qual distribueix els paràmetres entre els miralls i els volums, segons convingui.

5.2.6 Disseny del submòdul de volums

El diagrama de classes específic d'aquest submòdul és el de la Figura 5.14.

Ja hem dit abans que la funció d'aquest submòdul és gestionar tot el que fa referència als volums. Això vol dir que controla tots els paràmetres de visualització d'un volum als diferents miralls. Aquest submòdul, com altres, també consisteix en un parell de classes: **MagicMirrorsVolume** i **MagicMirrorsVolumeObserver**. La relació entre les dues és la mateixa que entre MagicMirrors i MagicMirrorsUpdater, és a dir, que la segona és una ajudant de la primera. Després ho explicarem amb més detall. Primer explicarem el per què d'aquest submòdul.

El mètode dels Miralls Màgics ha de permetre que un volum pugui tenir una funció de transferència diferent a cada mirall. Però amb VTK un volum (vtkVolume) només pot tenir una funció de transferència, ja que aquesta forma part del volum. Per solucionar aquest problema el que fem és crear un volum diferent per cada mirall, a més a més del volum central. Però llavors el que tenim són *n* volums independents i el que ens interessa és treballar-hi com si només en fos un. La solució a aquest nou problema és crear una nova classe que encapsuli tot el tractament de volums i executi les accions necessàries perquè tots els volums es comportin com un de sol. Això és exactament el que fa MagicMirrorsVolume.

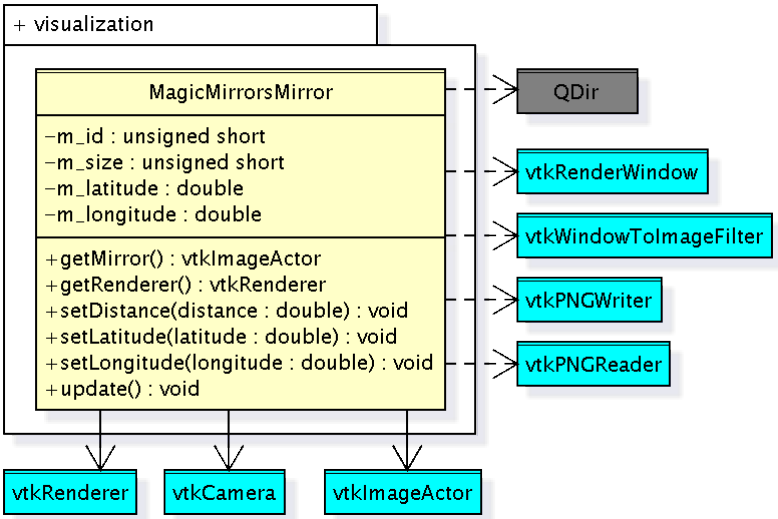


Figura 5.15: Diagrama de classes del submòdul de miralls dels Miralls Màgics.

MagicMirrorsVolume permet crear tants volums com siguin necessaris i accedir-hi fàcilment, només indicant l'índex. D'aquesta manera es pot obtenir o modificar la funció de transferència d'un vtkVolume concret. També és possible canviar el nombre de volums interns en qualsevol moment mitjançant el mètode setNumberOfVolumes(unsigned char).

L'altra tasca que realitza aquesta classe és mantenir sincronitzats tots els volums que emmagatzema, és a dir, mantenir-los alineats, a la mateixa posició i amb la mateixa rotació. Això fa semblin un sol volum de cara a l'usuari.

La sincronització es fa amb el mètode synchronize(), i aquest mètode és cridat per MagicMirrorsVolumeObserver. Aquesta última classe vigila el vtkVolume amb índex 0 (el que correspon al volum central) esperant l'esdeveniment de modificació. Per tant, quan es modifica el volum central (quan es mou o gira) els altres volums es modifiquen de la mateixa manera.

5.2.7 Disseny del submòdul de miralls

El diagrama de classes específic d'aquest submòdul és el de la Figura 5.15.

Aquest últim submòdul dels Miralls Màgics té la finalitat d'encapsular el tractament d'un mirall: la creació, el posicionament i les actualitzacions. L'única classe que en forma part és **MagicMirrorsMirror**.

La classe disposa de mètodes per obtenir l'objecte que fa de mirall (vtkImageActor) i el *renderer* del mirall. El primer serveix perquè MagicMirrors pugui afegir el vtkImageActor al *renderer* principal i per tant es vegi el mirall a la finestra de visualització principal. El segon serveix perquè MagicMirrors afegeixi els volums al *renderer* del mirall (o perquè els esborri). També disposa de mètodes per assignar la posició (distància, latitud i longitud).

Per acabar, té el mètode update() que actualitza el mirall. Per actualitzar el mirall crea una finestra de visualització (vtkRenderWindow), converteix la seva visualització en una imatge mitjançant un vtkWindowToImageFilter, escriu aquesta imatge en un fitxer temporal mitjançant vtkPNGWriter, llegeix la imatge del fitxer temporal mitjançant vtkPNGReader, i enganxa aquesta imatge al vtkImageActor. Actualment no es pot enganxar la imatge directament al vtkImageActor sense escriure-la primer en un fitxer, i per això es realitza tot aquest procés. El mètode update() és cridat per MagicMirrors quan s'han d'actualitzar els miralls.

5.3 Disseny del mòdul de la selecció del punt de vista

Com hem dit abans, aquest mòdul conté totes les classes que estan dissenyades específicament per implementar la solució al problema de la selecció del punt de vista òptim. El seu diagrama de classes general és el que es pot veure a la Figura 5.16.

És un diagrama molt semblant al dels Miralls Màgics, on les diferències principals són que només té un formulari d'introducció de paràmetres (ja que només tenim un volum i la visualització és la mateixa a tots els plans) i que té més classes al paquet visualization.

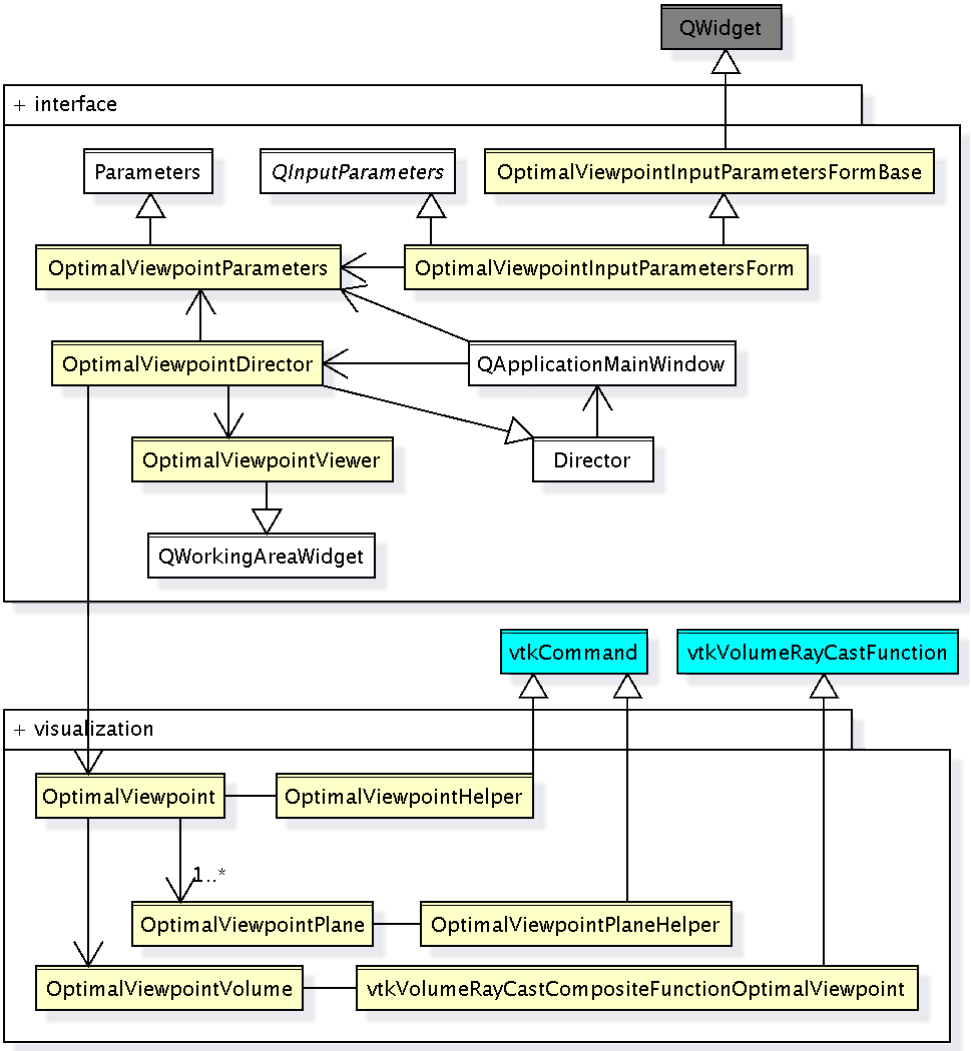


Figura 5.16: Diagrama de classes del mòdul de la selecció del punt de vista.

Veiem ara en detall cada submòdul.

5.3.1 Disseny del submòdul d'introducció de paràmetres

El diagrama de classes específic d'aquest submòdul és el de la Figura 5.17.

Com ja hem dit, les classes d'aquest submòdul permeten introduir tots els paràmetres d'entrada d'aquest mètode de selecció del punt de vista òptim. Són un parell de classes: **OptimalViewpointInputParametersFormBase** i **OptimalViewpointInputParametersForm**.

Concretament, els paràmetres que permeten introduir aquestes classes són:

- **Volums a visualitzar:** és l'identificador (Identifier) del volum que es visualitzarà.

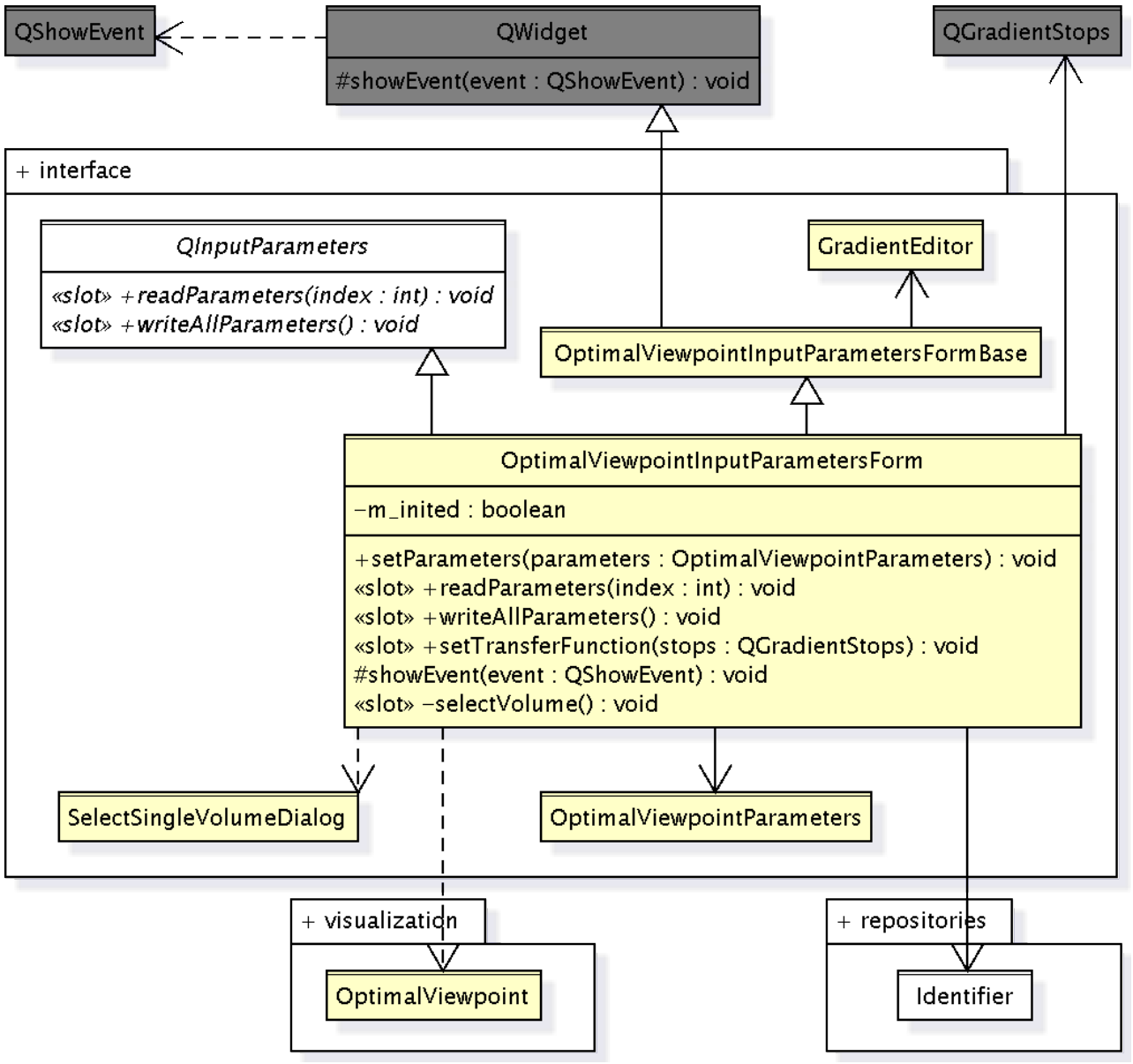


Figura 5.17: Diagrama de classes del submòdul d'introducció de paràmetres de la selecció del punt de vista.

- **Paràmetres de la segmentació:** són tots els paràmetres del mètode de segmentació, el qual inclou el càlcul de l'*excess entropy* des d'un punt de vista fix:
  - Nombre d'iteracions de l'algorisme genètic
  - Soroll per a l'algorisme genètic
  - Nombre de clústers
  - Longitud de bloc (per calcular l'*excess entropy*)
  - Distància entre raigs llançats (per calcular l'*excess entropy*)

- Distància entre mostres d'un raig (per calcular l'*excess entropy*)

- **Paràmetres de la visualització:** són tots els paràmetres que afecten a la visualització, incloent el càlcul de l'*excess entropy* dels punts de vista (plans):
  - Nombre de punts de vista o plans: permet triar entre 4, 6, 8, 12 o 20 plans distribuïts uniformement al voltant del volum
  - Ombrejat: és un booleà que indica si el volum es pintarà amb ombrejat
  - Longitud de bloc (per calcular l'*excess entropy*)
  - Distància entre raigs llançats (per calcular l'*excess entropy*)
  - Distància entre mostres d'un raig (per calcular l'*excess entropy*)
  - Funció de transferència

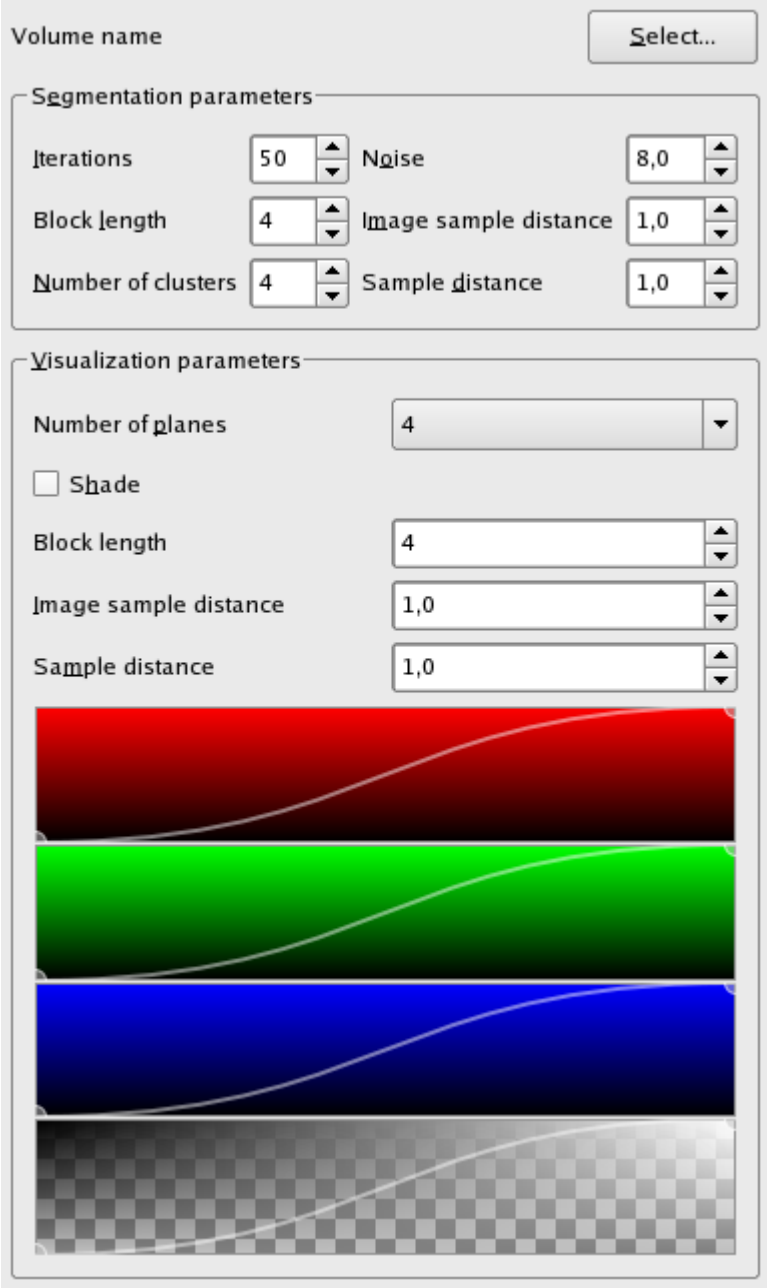


Figura 5.18: OptimalViewpointInputParametersForm.

Els paràmetres de l'*excess entropy* per fer la segmentació i de cada punt de vista no han de ser necessàriament iguals, i és per això que aquests paràmetres apareixen dues vegades.

La classe base un altre cop serveix només per definir el formulari i la classe filla per fer-ne la implementació. En aquest cas també es fa servir el GradientEditor per definir la funció de transferència, exactament igual que a MagicMirrorsMirrorVolumeInputParametersFormBase i MagicMirrorsMirrorVolumeInputParametersForm (fins i tot amb les mateixes connexions de *signals i slots* i la reimplementació de showEvent(QShowEvent*) per inicialitzar el GradientEditor).

OptimalViewpointInputParametersForm també s'encarrega de mostrar el diàleg per seleccionar un volum (SelectSingleVolumeDialog), al mètode selectVolume(), que és cridat quan l'usuari prem el botó "Select...".

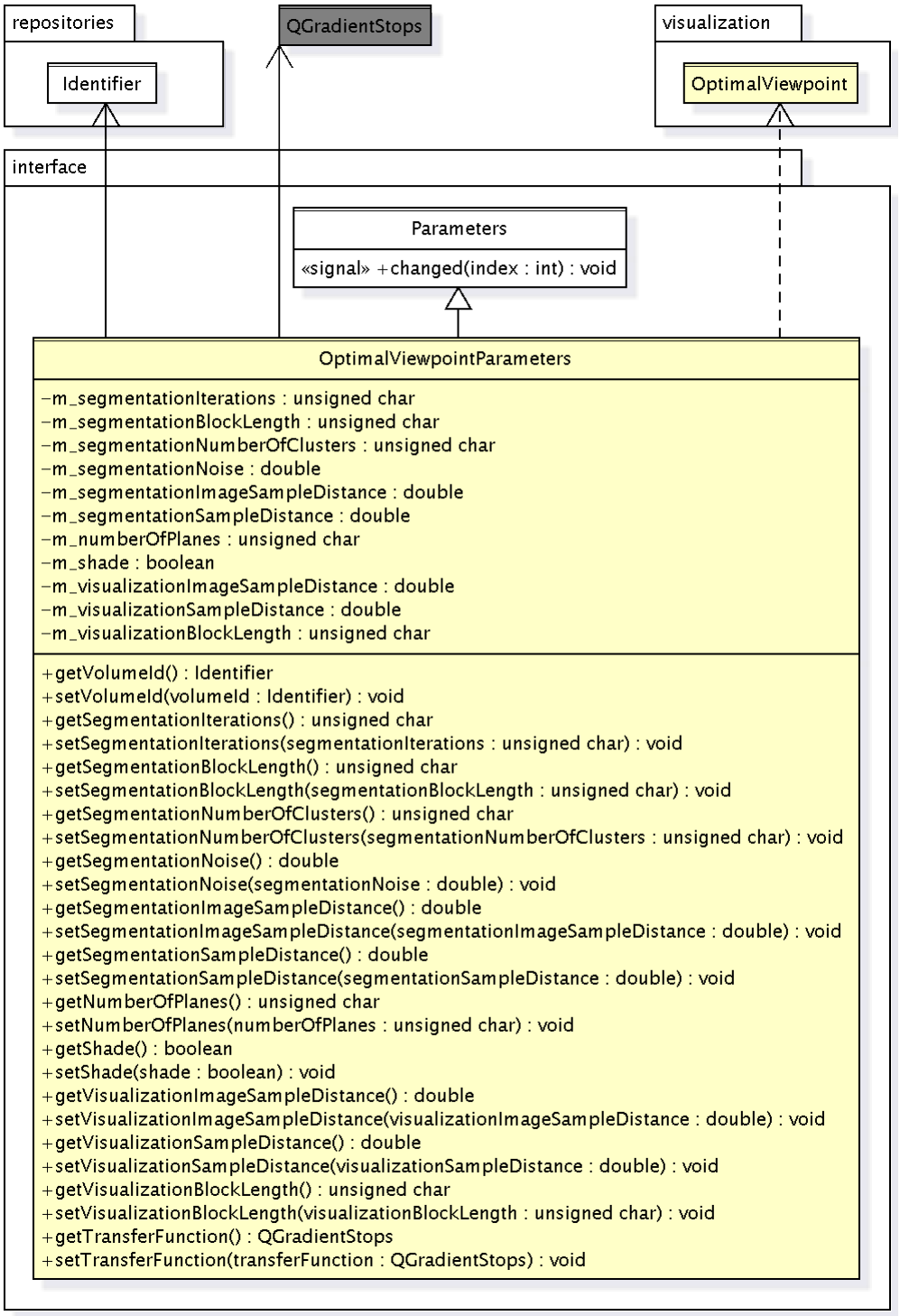


Figura 5.19: Diagrama de classes del submòdul d'encapsulament de paràmetres de la selecció del punt de vista.

Per acabar, aquesta classe també té els dos *slots* heretats de QInputParameters, amb els quals es comunica amb OptimalViewpointParameters per llegir o escriure qualsevol paràmetre. Aquests *slots* són readParameter(int) i writeAllParameters().

5.3.2 Disseny del submòdul d'encapsulament de paràmetres

El diagrama de classes específic d'aquest submòdul és el de la Figura 5.19.

Recordem que la funció d'aquest submòdul és guardar tots els paràmetres d'entrada del mètode de selecció del punt de vista òptim per mantenir-los tots en un sol lloc. L'única classe d'aquest mòdul és **OptimalViewpointParameters**, que és una subclasse de Parameters.

La classe defineix una enumeració amb els noms dels paràmetres, OptimalViewpointParametersNames, amb els elements següents: VolumeId, SegmentationIterations, SegmentationBlockLength, SegmentationNumberOfClusters, SegmentationNoise, SegmentationImageSampleDistance, SegmentationSampleDistance, NumberOfPlanes, Shade, VisualizationBlockLength, VisualizationImageSampleDistance, VisualizationSampleDistance, TransferFunction.

Cada paràmetre està guardat en un atribut diferent i per cada paràmetre hi ha els mètodes *set* i *get* corresponents. Els *sets* emeten el *signal* changed(int) heretat de Parameters, amb el nom del paràmetre (agafat d'OptimalViewpointParametersNames) com a argument. Aquest *signal* serà recollit per OptimalViewpointInputParametersForm i li dirà que ha de llegir el paràmetre indicat.

Aquest submòdul no té res més a destacar perquè només es tracta del magatzem de paràmetres del mètode.

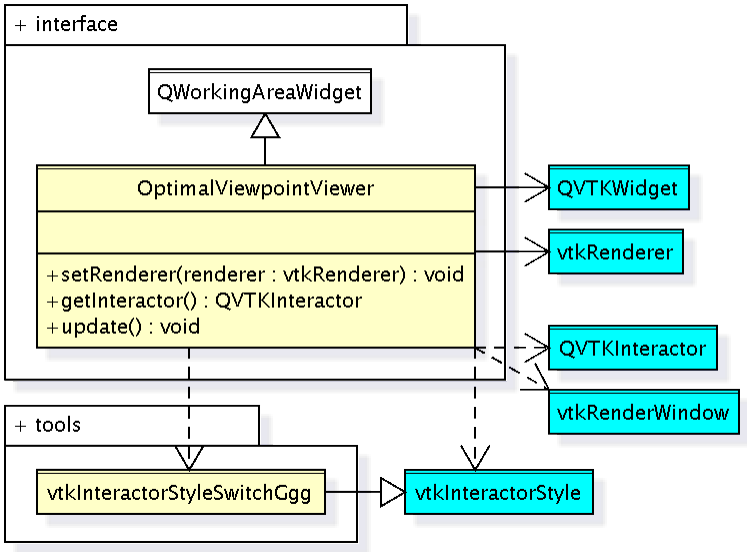


Figura 5.20: Diagrama de classes del submòdul de la finestra de visualització principal de la selecció del punt de vista.

5.3.3 Disseny del submòdul de la finestra de visualització principal

El diagrama de classes específic d'aquest submòdul és el de la Figura 5.20.

Com ja hem dit, aquest submòdul fa referència a la finestra de visualització principal, on es veu el model amb els plans al voltant. L'única classe d'aquest mòdul és **OptimalViewpointViewer**.

Aquesta és classe hereta de QWorkingAreaWidget perquè la finestra aparegui a l'àrea de treball de l'StarViewer.

La classe és mol senzilla, ja que només es tracta d'una finestra. Té com a únics atributs un QVTKWidget, on es veurà la visualització, i un vtkRenderer, que és el que s'encarrega de dibuixar la visualització.

Té un mètode per assignar-li el *renderer* que ha de fer servir, un per obtenir l'*interactor* (l'objecte que permet interaccionar directament amb la visualització) i un per forçar l'actualització de la visualització.

Podem destacar com a particularitat que fa servir la classe vtkInteractorStyleSwitchGgg, del paquet tools, en comptes del vtkInteractorStyleSwitch de VTK. El motiu és que la versió GGG, que és una modificació de l'original, té un temps de resposta més petit quan es vol interaccionar amb el model, però això ho explicarem amb detall quan expliquem el disseny del mòdul de les classes compartides.

A banda d'això, no hi ha res més a destacar en aquest submòdul.

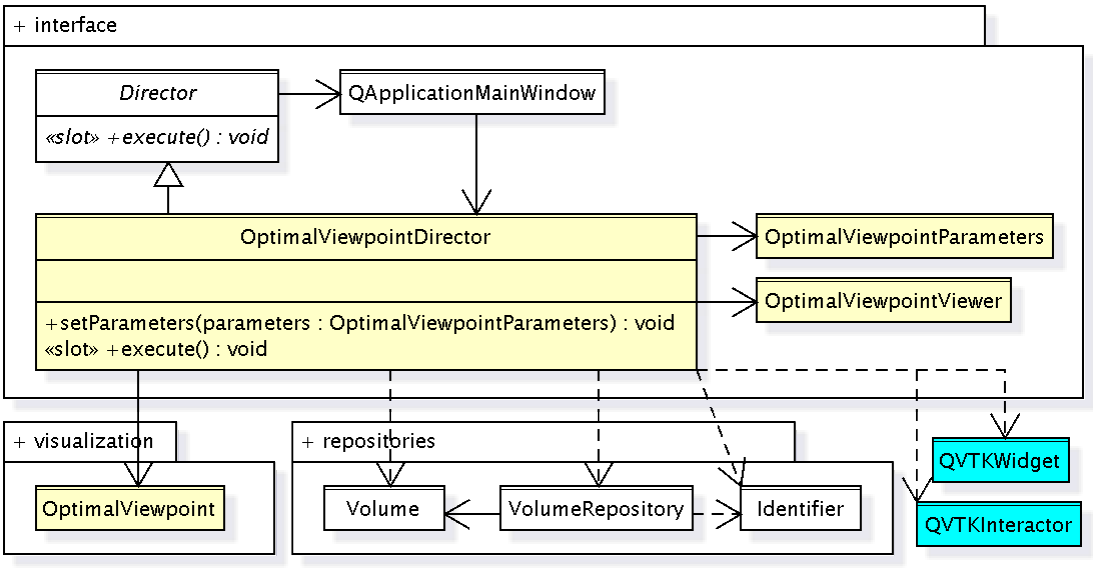


Figura 5.21: Diagrama de classes del submòdul d'enllaç de la interfície amb les classes de control de la selecció del punt de vista.

5.3.4 Disseny del submòdul d'enllaç de la interfície amb les classes de control

El diagrama de classes específic d'aquest submòdul és el de la Figura 5.21.

Ja hem dit abans que aquest submòdul s'encarrega de relacionar les classes que formen part de la interfície amb les classes de control, és a dir, les que fan la visualització i calculen l'*excess entropy* de cada punt de vista, i que es troben al paquet visualization. L'única classe que en forma part és **OptimalViewpointDirector**.

Aquesta classe és filla de Director i d'aquest hereta l'*slot* execute(), el qual ha d'implementar perquè executi el mètode. També disposa d'un mètode perquè li assignin l'objecte de paràmetres.

Per poder realitzar la seva feina de coordinació, aquesta classe manté punters al mètode (OptimalViewpoint), als paràmetres (OptimalViewpointParameters) i a la finestra de visualització principal (OptimalViewpointViewer), així com un punter a la finestra principal de l'aplicació (QApplicationMainWindow) heretat de Director.

La primera vegada que es crida l'execute(), crea el mètode, i també crea el visor i l'afegeix a l'àrea de treball. També assigna el volum seleccionat al mètode, agafant-lo del repositori mitjançant l'identificador. Un cop fet això crida el mètode de segmentació amb els paràmetres de la segmentació.

La resta de tasques les fa cada vegada, i consisteixen en passar els paràmetres de visualització al mètode, cridar el mètode per actualitzar els plans i cridar el mètode del visor per actualitzar la visualització. Si s'ha calculat l'*excess entropy* de cada punt de vista, en mostra els resultats.

Això és tot el que fa aquest submòdul.

5.3.5 Disseny del submòdul del control principal de la visualització

El diagrama de classes específic d'aquest submòdul és el de la Figura 5.22.

Aquest submòdul està format per les classes que controlen la visualització principal del mètode de la selecció del punt de vista òptim. Aquestes classes són **OptimalViewpoint** i **OptimalViewpointHelper**.

OptimalViewpoint és la classe principal, la que fa realment la feina. OptimalViewpointHelper és un ajudant d'OptimalViewpoint, que crida alguns dels seus mètodes d'actualització quan cal.

OptimalViewpoint guarda un volum (OptimalViewpointVolume) i un conjunt de plans (OptimalViewpointPlane), i s'encarrega de distribuir els paràmetres de segmentació i de visualització rebuts entre el volum i els plans. Evidentment, també fa que es visualitzin a la finestra principal tant el volum com els plans.

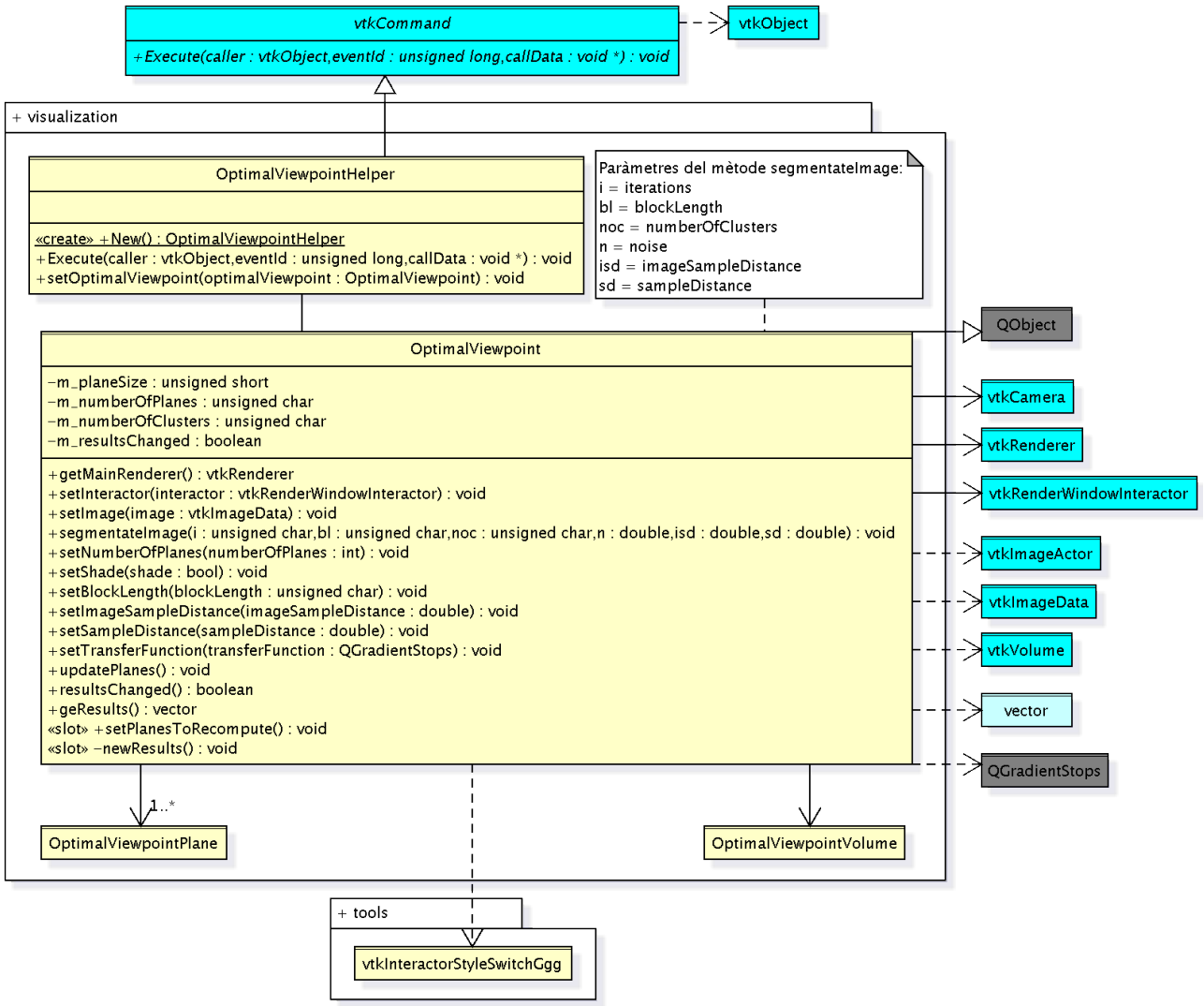


Figura 5.22: Diagrama de classes del submòdul del control principal de la visualització de la selecció del punt de vista.

Igual que en el cas dels Miralls Màgics, OptimalViewpoint s'encarrega de crear el *renderer* principal i el retorna amb el mètode getMainRenderer(). També afegeix i treu objectes del *renderer* per fer que es vegin o no. També cal que el volum sigui l'últim objecte afegit.

També com en el cas dels Miralls Màgics, la classe OptimalViewpoint necessita accedir a l'*interactor* per afegir-hi un observador, l'OptimalViewpointHelper. L'OptimalViewpointHelper vigila l'*interactor* esperant l'esdeveniment de fi d'interacció, i quan l'*interactor* el genera es crida el mètode Execute(...) d'OptimalViewpointHelper, el qual crida el mètode setPlanesToRecompute() d'OptimalViewpoint. Aquest mètode diu als plans que han de tornar a calcular l'*excess entropy*, i en aquest cas cal fer-ho perquè l'usuari ha girat o mogut el volum, i per tant la vista ha canviat. Amb tot això quan l'usuari acabi una interacció amb el model els plans calcularan la nova *excess entropy* quan l'usuari premi el botó "Apply" per actualitzar-los.

OptimalViewpoint també redueix els valors de propietats del model a un rang entre 0 i 255 pels mateixos motius que MagicMirrors, a més a més que és més fàcil aplicar un mètode de segmentació quan el rang de valors inicial és fix. També s'agafa la diagonal del model com a mida de pla.

Un dels mètodes importants d'OptimalViewpoint és segmentateImage(...), que rep els paràmetres necessaris per realitzar la segmentació i realitza els passos necessaris per poder-la dur a terme. Primer assigna la distància entre raigs i la distància entre mostres al volum. Crea un pla temporal que servirà per calcular l'*excess entropy* del mètode de segmentació i li assigna els paràmetres que necessita (longitud de bloc i nombre de clústers). Aquest pla estarà situat sobre l'eix Z. Després fa diverses connexions de *signals i slots* entre el el volum i el pla, que serviran com a mecanisme de comunicació entre els dos. Quan està tot preparat crida el mètode segmentateVolume(...) del volum passant-li el nombre d'iteracions, el nombre de clústers i el soroll. Quan acaba l'execució del mètode el volum ja està segmentat i es destrueix el pla temporal.

Un altre mètode important és el setNumberOfPlanes(unsigned char), que serveix per triar el nombre de plans. Aquest mètode crea o destrueix els plans que calgui i els distribueix uniformement al voltant del volum. El nombre de plans es pot canviar en qualsevol moment.

OptimalViewpoint també té tots els mètodes per assignar els paràmetres de la visualització, i els assigna a l'objecte que calgui (volum o plans).

Hi ha el mètode updatePlanes() que s'encarrega de cridar el mètode update() de cada pla perquè s'actualitzin.

Pel que fa a l'accés als resultats del càlcul de l'*excess entropy* de cada pla, hi ha el mètode resultsChanged(), que retorna cert quan s'ha recalculat l'*excess entropy*, i getResults(), que retorna un vector amb els resultats de cada pla.

Ja per acabar, hi ha dos *slots*. El primer és públic i és setPlanesToRecompute(). A part de ser cridat per OptimalViewpointHelper quan l'usuari interacciona amb el model, també pot ser executat quan el volum emet el *signal* samplingChanged(), que s'emet quan canvia la distància entre raigs o la distància entre mostres. Si canvia algun d'aquests valors també canvia l'*excess entropy* i per tant els plans l'han de tornar a calcular. L'altre *slot* és privat i és newResults(), i s'executarà quan els plans acabin de calcular l'*excess entropy*. Serveix per avisar OptimalViewpoint que els resultats han canviat.

5.3.6 Disseny del submòdul de volums

El diagrama de classes específic d'aquest submòdul és el de la Figura 5.23.

Ja hem dit que abans que aquest submòdul s'encarrega de gestionar tot el que fa referència als volums. Això vol dir que controla tots els paràmetres de visualització del volum i el segmenta per crear un model simplificat. Les dues classes que en formen són **OptimalViewpointVolume** i **vtkVolumeRayCastCompositeFunctionOptimalViewpoint**.

Per millorar el rendiment de mètode de visualització OptimalViewpointVolume guarda en realitat dos volums. Un és per fer la visualització del model central i aquesta visualització és amb les classes normals de VTK. L'altre volum és el que es fa servir per la visualització als plans i fa servir les

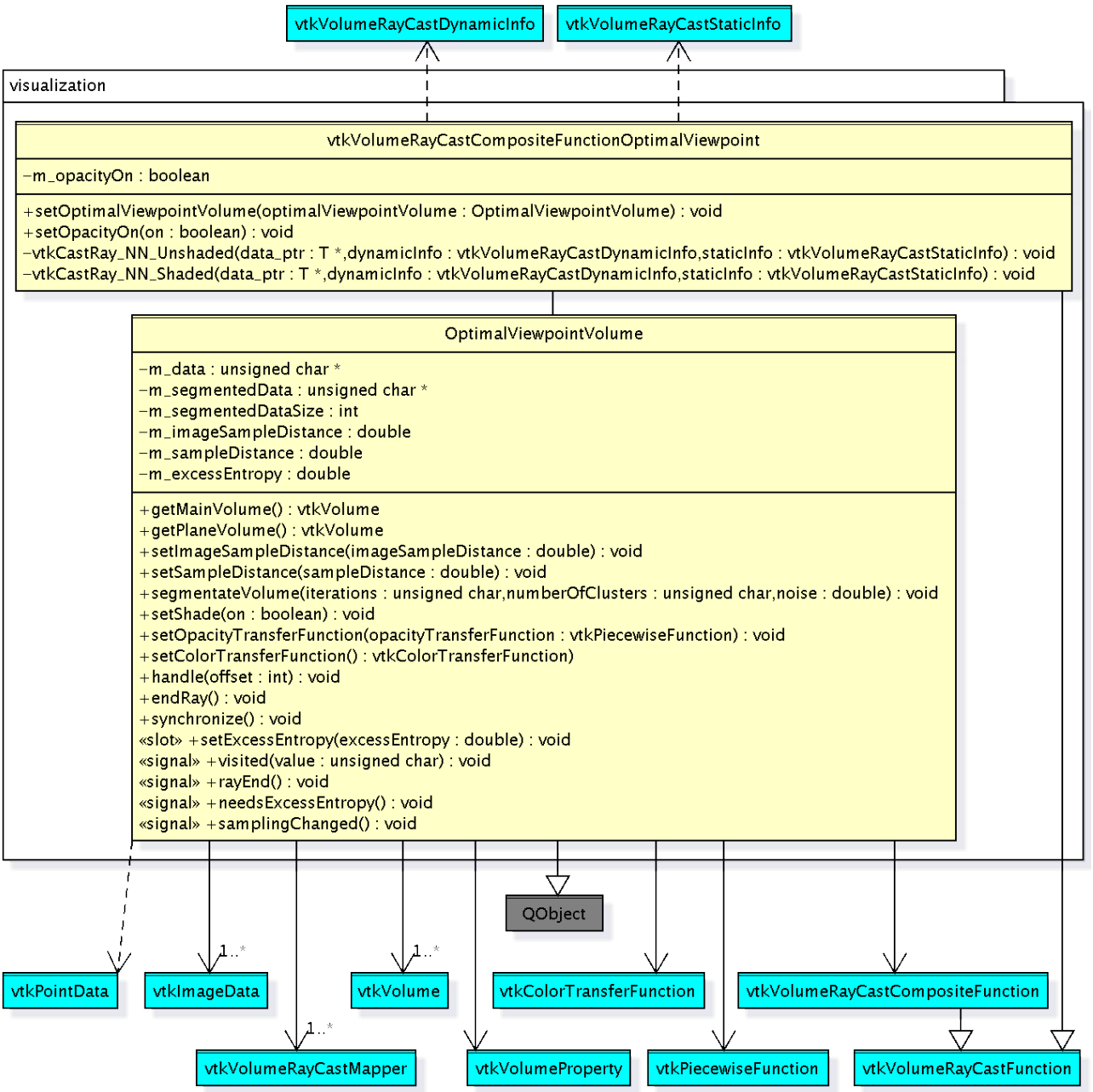


Figura 5.23: Diagrama de classes del submòdul de volums de la selecció del punt de vista.

classes normals de VTK, excepte una: en lloc de fer servir vtkVolumeRayCastCompositeFunction fa servir vtkVolumeRayCastCompositeFunctionOptimalViewpoint, que és una versió modificada de l'original.

Aquesta nova classe manté un punter a OptimalViewpointVolume per cridar alguns dels seus mètodes quan fa la visualització. Per cada raig llançat en el procés de *ray casting* es crida el mètode CastRay(...) de vtkVolumeRayCastCompositeFunctionOptimalViewpoint, i aquest mètode crida un mètode vtkCastRay_*_*(...) segons el tipus d'interpolació i segons si hi ha ombrejat o no. En la nostra implementació sempre hi ha la interpolació per defecte (veí més proper) i per tant només depèn de l'ombrejat. Llavors s'executarà vtkCastRay_NN_Unshaded(...) o vtkCastRay_NN_Shaded(...) segons si hi ha ombrejat o no. Tots dos tenen les mateixes modificacions respecte l'original:

- Només tenen en compte l'opacitat del volum per determinar fins a quin punt arriben del raig si m_opacityOn és cert. És a dir, si m_opacityOn és cert el raig només arriba fins al punt on ha acumulat una opacitat molt alta o fins al final; si m_opacityOn és fals el raig sempre arriba fins al final. Això és útil perquè quan s'ha de calcular l'*excess entropy* és necessari que el raig arribi sempre fins al final per agafar totes les mostres possibles, però es pot accelerar el procés de visualització fent un raig més curt quan no s'està calculant l'*excess entropy*.

- Per cada mostra agafada criden el mètode handle(int) d'OptimalViewpointVolume passant com a paràmetre un desplaçament respecte el l'inici del punter a les dades. L'objectiu d'això és que OptimalViewpointVolume tracti la mostra corresponent del model segmentat per calcular l'*excess entropy*.

- Quan acaben el raig criden el mètode endRay() d'OptimalViewpointVolume. Això també té utilitat per al càlcul de l'*excess entropy*.

L'augment de rendiment que hem comentat abans per justificar els dos volums separats ve del fet que pel volum central no cal calcular mai l'*excess entropy*, i per tant no cal que es vagin cridant mètodes per cada vòxel visitat, ja que només afegirien una càrrega extra innecessària, i tampoc no cal traçar els raigs fins al final. També permet que la distància entre raigs i la distància entre mostres pel volum central siguin automàtiques; per exemple, si el model està quiet aquestes distàncies són petites per dibuixar el volum amb tots els detalls; en canvi, mentre l'usuari està manipulant el volum aquestes distàncies són grans i el model es dibuixa més ràpid però amb menys detalls. Per tant, és millor que el volum central sigui un volum diferent del dels plans i faci servir les classes estàndards de VTK.

D'altra banda, en tenir dos volums cal també un mètode per mantenir-los sincronitzats en les rotacions i els moviments. Aquest mètode és synchronize().

OptimalViewpointVolume té els mètodes per assignar la distància entre raigs i la distància entre mostres. Si hi ha alguna variació en alguna de les distàncies s'emet el *signal* samplingChanged() per avisar OptimalViewpointPlane perquè torni a calcular l'*excess entropy*.

També té els mètodes necessaris per assignar si es fa ombrejat o no, la funció de transferència d'opacitat i la funció de transferència de color.

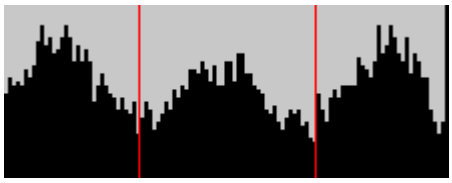


Figura 5.24: Exemple de segmentació d'una imatge en 3 clústers. Les línies vermelles són els límits.

Ja hem dit que és aquí on es fa la segmentació. Concretament és al mètode segmentateVolume(...), que rep com a paràmetres el nombre d'iteracions de l'algorisme genètic, el nombre de clústers i el soroll. El mètode de segmentació segmenta el volum en el nombre de clústers[5] que li diem, i ho fa aplicant 3 algorismes diferents. La idea és anar movent els límits de la segmentació fins a trobar els que s'ajustin millor al model per al nombre de clústers donat. Els límits de la segmentació són els valors de propietat on es canvia de clúster. Per exemple, si tinguéssim dos clústers amb un límit al valor 5 voldria dir que el primer clúster va del 0 al 5 i el segon del 6 al 255. A la Figura 5.24 en veiem un exemple gràfic amb 3 clústers i dos límits.

5 Un clúster fa referència a tot un conjunt de vòxels que formen una mateixa estructura perquè tenen un valor de propietat semblant. Un clúster no ha de ser necessàriament una regió contínua.

El mètode comença inicialitzant els límits amb uns valors distribuïts uniformement, per crear tots els clústers amb la mateixa llargada. A partir d'aquests límits inicials es van aplicant els algorismes un darrere l'altre, i tots modifiquen els límits.

El primer és un algorisme genètic que fa el nombre d'iteracions indicat al paràmetre. A cada iteració:

- Ordena els límits actuals.
- Segmenta la imatge amb els límits ordenats.
- Demana l'*excess entropy* de la imatge segmentada emetent el *signal* needsExcessEntropy(), que serà recollit per l'*slot* updateAndRecompute() del pla temporal que hem comentat abans. Aquest farà els càlculs i al final OptimalViewpointVolume rebrà el resultat a l'*slot* setExcessEntropy(double). Aquests càlculs es fan durant el *ray casting* per dibuixar el volum simplificat, i per tant passa per vtkVolumeRayCastCompositeFunctionOptimalViewpoint, que crida mètodes d'OptimalViewpointVolume, concretament:
  - handle(int): Emet el *signal* visited(unsigned char) passant com a paràmetre el valor de propietat corresponent al model segmentat. Aquest *signal* serà recollit per l'*slot* compute(unsigned char) d'OptimalViewpointPlane.
  - endRay(): Emet el *signal* rayEnd() que serà recollit per l'*slot* endLBlock() d'OptimalViewpointPlane.
- Si el nou valor de l'*excess entropy* és el màxim fins ara, guarda els límits ordenats com a màxims.
- Calcula els pròxims límits sumant a cada límit màxim el soroll multiplicat per un valor aleatori entre -1 i 1.

El segon algorisme és d'intercanvi. Fa dues iteracions i en cadascuna d'elles i per cada límit:

- Actualitza els límits actuals amb els màxims.
- Itera sobre un valor k2 que va de 0 a 256 (exclòs) avançant de 8 en 8, i per cada valor de k2:
  - Canvia el límit per k2.
  - Ordena els límits actuals.
  - Segmenta la imatge amb els límits ordenats.
  - Demana l'*excess entropy*.
  - Si el nou valor de l'*excess entropy* és màxim, guarda els límits ordenats com a màxims.

El tercer i últim algorisme és d'afinament. Mentre es produeixin canvis als límits màxims i per cada límit:

- Actualitza els límits actuals amb els màxims.
- Resta 1 al límit.
- Ordena els límits actuals.
- Segmenta la imatge amb els límits ordenats.
- Demana l'*excess entropy*.
- Si el nou valor de l'*excess entropy* és màxim, guarda els límits ordenats com a màxims.
- Suma 2 al límit (per tant queda com si hagués sumat 1 a l'original).
- Ordena els límits actuals.
- Segmenta la imatge amb els límits ordenats.
- Demana l'*excess entropy*.
- Si el nou valor de l'*excess entropy* és màxim, guarda els límits ordenats com a màxims.

Quan acaba el tercer algorisme el mètode fa la segmentació final amb els límits màxims ordenats i acaba. La imatge segmentada és la que es visualitzarà als plans.

Després de segmentar el model, el mètode defineix una funció de transferència ajustada al model. La manera de fer-ho consisteix a definir una funció de transferència amb tants trams com clústers té la imatge segmentada. Cada tram va entre dos límits de segmentació, amb l'excepció del primer, que va de 0 al primer límit, i l'últim, que va de l'últim límit a 255. Per exemple, si segmentem un model en tres clústers i els límits són 25 i 55, llavors hi haurà un tram de 0 a 25, un de 26 a 55 i un de 56 a 255.

A cada tram se li assigna un únic valor RGBA calculat aleatòriament, amb l'excepció del primer tram, que té una A fixa amb un valor molt petit. Ho fem d'aquesta manera perquè el primer clúster normalment correspon a l'espai buit al voltant del model i per tant no conté informació útil.

La funció de transferència calculada es propaga fins a OptimalViewpoint mitjançant signals i slots. OptimalViewpointDirector obté la funció de transferència d'OptimalViewpoint i l'assigna a OptimalViewpointParameters. D'aquesta manera la nova funció de transferència s'aplicarà quan es visualitzi el volum.

5.3.7 Disseny del submòdul de plans

El diagrama de classes específic d'aquest submòdul és el de la Figura 5.25.

L'últim submòdul del mòdul de la selecció del punt de vista té la finalitat d'encapsular el tractament d'un pla: la creació, el posicionament, les actualitzacions i el càlcul de l'*excess entropy* des del punt de vista del pla. Les dues classes que en formen part són **OptimalViewpointPlane** i **OptimalViewpointPlaneHelper**.

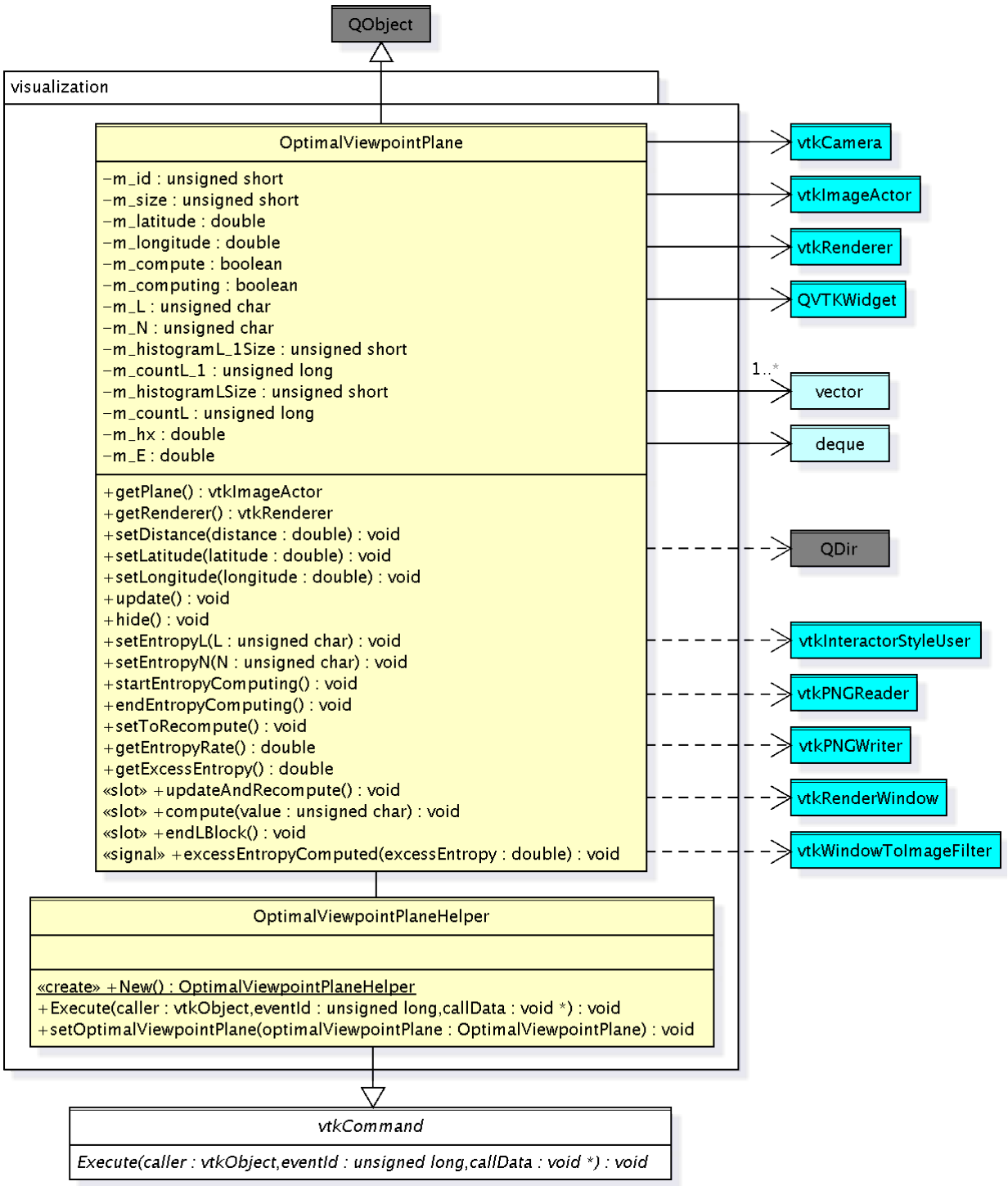


Figura 5.25: Diagrama de classes del submòdul de plans de la selecció del punt de vista.

La classe principal disposa de mètodes per obtenir l'objecte que fa de pla (vtkImageActor) i el *renderer* del pla. El primer serveix perquè OptimalViewpoint pugui afegir el vtkImageActor al *renderer* principal i per tant es vegi el pla a la finestra de visualització principal. El segon serveix perquè OptimalViewpoint afegeixi el volum al *renderer* del pla (o perquè l'esborri). També disposa de mètodes per assignar la posició (distància, latitud i longitud). Fins aquí és com MagicMirrorsMirror.

També, com MagicMirrorsMirror té el mètode update() que actualitza el mirall. El procés és el mateix però amb la diferència que la finestra no és temporal sinó que és permanent i és un objecte QVTKWidget. El mètode update() és cridat per OptimalViewpoint quan s'han d'actualitzar els plans i per l'*slot* updateAndRecompute() d'OptimalViewpointPlane.

Però el pla també ha de calcular l'*excess entropy* des del seu propi punt de vista, i per tant apareixen tot un conjunt d'atributs i mètodes nous, a més de la classe ajudant.

Per començar, com que la finestra és permanent i s'hauria d'amagar quan es redueix el nombre de plans (per estalviar-se destruir l'objecte i tornar-lo a crear si s'augmenta el nombre de plans més tard) apareix el mètode hide(). Aquest mètode només serveix per amagar la finestra. No hi ha cap mètode per mostrar-la perquè això ja ho fa el mètode update().

També hi ha mètodes per assignar els dos paràmetres de l'*excess entropy*: L i N. L és la longitud de bloc i N el nombre de possibles valors, que és igual al nombre de clústers del model segmentat.

Per al càlcul de l'*excess entropy* hi ha els atributs següents:

- m_compute: només es començarà a calcular l'*excess entropy* quan aquest atribut sigui cert
- m_computing: és cert mentre dura el càlcul de l'*excess entropy*; el mètode compute(unsigned char) només té efecte quan aquest atribut i l'anterior són certs
- m_L: el paràmetre L de l'*excess entropy*
- m_N: el paràmetre N de l'*excess entropy*
- m_histogramL_1Size: és la mida l'histograma dels blocs de longitud $L - 1$, és a dir, quants possibles blocs de longitud $L - 1$ hi ha
- m_histogramL_1: és l'histograma dels blocs de longitud $L - 1$; és un vector
- m_countL_1: és el nombre de valors de l'histograma dels blocs de longitud $L - 1$
- m_histogramLSize: és la mida l'histograma dels blocs de longitud L, és a dir, quants possibles blocs de longitud L hi ha
- m_histogramL: és l'histograma dels blocs de longitud L; és un vector
- m_countL: és el nombre de valors de l'histograma dels blocs de longitud L
- m_lastLValues: són els últims L valors visitats pel raig actual del *ray casting*; és una *deque*
- m_hx: resultat de l'*entropy rate*
- m_E: resultat de l'*excess entropy*

El càlcul de l'*excess entropy* s'inicia automàticament quan s'actualitza el pla gràcies a OptimalViewpointPlaneHelper. Aquest últim observa el *renderer* del pla esperant els esdeveniments

d'inici i final de *rendering*. Quan comença el *rendering* crida el mètode startExcessEntropyComputing(), i quan acaba el *rendering* crida el mètode endExcessEntropyComputing().

startExcessEntropyComputing() inicialitza els histogrames i els seus comptadors i també la *deque* dels últims L valors, i posa m_computing a cert. Tot això només si m_compute és cert.

endExcessEntropyComputing() fa el procés contrari. Posa m_compute i m_computing a fals i fa el càlcul final de l'*excess entropy* amb la informació dels histogrames, aplicant les fórmules que hem vist al capítol dels fonaments teòrics. Amb els histogrames es calculen les entropies dels blocs de longitud L i $L - 1$, amb aquestes entropies es calcula l'*entropy rate*, i a partir d'aquest l'*excess entropy*. Finalment destrueix els histogrames i la *deque* i emet el *signal* excessEntropyComputed(double), passant com a paràmetre el valor de l'*excess entropy* guardat a m_E. Aquest *signal* serà rebut per l'*slot* setExcessEntropy(double) d'OptimalViewpointVolume. Tot això només si m_compute és cert.

També hi ha el mètode setToRecompute(), que es limita a posar m_compute a cert perquè es recalculi l'*excess entropy* quan s'actualitzi el pla.

Per acabar amb els mètodes normals, hi ha getEntropyRate() i getExcessEntropy(), per obtenir els resultats dels dos càlculs.

El primer *slot* que hi ha és updateAndRecompute(), que l'únic que fa és cridar setToRecompute() i update(). Aquest *slot* s'activarà quan OptimalViewpointVolume necessiti l'*excess entropy*.

El segon *slot* és compute(unsiged char). Serveix per anar omplint els histogrames amb els valors que es van visitant. Primer de tot cal recordar que només té efecte si m_compute i m_computing són certs. Si és així es guarda el valor al davant de la *deque* dels últims valors. Si la *deque* ja té $L - 1$ o més valors, és a dir, que tenim un bloc de longitud $L - 1$ o més, s'incrementa el valor de la casella de l'histograma $L - 1$ que corresponent al bloc. El bloc són els $L - 1$ primers valors de la *deque*. També s'incrementa el comptador d'aquell histograma. Després, si la *deque* té L valors es fa el mateix respecte als blocs de longitud L, i a més es treu l'últim valor de la *deque*. La posició i de l'histograma que correspon a un bloc de longitud L $x_0x_1...x_L$, on els valors van del més recent al més antic ve donada per la fórmula $i = \sum_{j=0}^{L} x_j \cdot N^j$. És a dir, es considera el bloc $x_0x_1...x_L$ com un nombre d'L xifres en base N i es passa a decimal. Aquest *slot* s'executa cada vegada que es visita un vòxel en durant el *ray casting*.

L'últim *slot* és endLBlock(), que buida la *deque*. S'executa cada vegada que s'acaba un raig del *ray casting*, ja que això significa la fi del bloc.

5.4 Disseny del mòdul de les classes compartides

Com hem dit abans, aquest mòdul conté les classes que no estan dissenyades específicament per a cap dels dos mètodes anteriors, sinó que són útils per a coses més generals. El seu diagrama de classes general és el que es pot veure a la Figura 5.26.

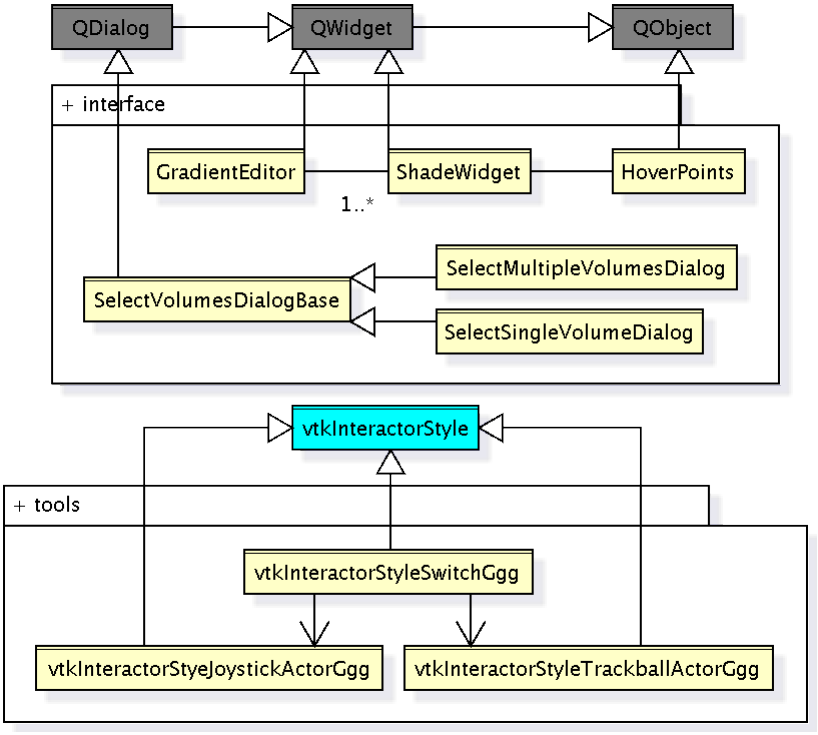


Figura 5.26: Diagrama de classes del mòdul de les classes compartides.

Aquest diagrama no té res a veure amb els dels altres mòduls, ja que no és d'un mètode de registre, segmentació o visualització. Es poden distingir clarament els tres submòduls, amb tres classes cadascun. Veiem en detall cadascun d'ells.

5.4.1 Disseny del submòdul de definició de funcions de transferència

El diagrama de classes específic d'aquest submòdul és el de la Figura 5.27.

Com ja hem dit abans, aquest submòdul està format per les classes de la interfície que permeten definir una funció de transferència. Aquestes classes són **HoverPoints**, **ShadeWidget** i **GradientEditor**.

Aquestes classes estan basades en les que es poden trobar a les demos de Qt 4, però les hem adaptat a aquest projecte creant un fitxer .h i un .cpp per cadascuna (a la demo ShadeWidget i GradientEditor estan juntes) i fent les modificacions pertinents perquè funcionessin d'aquesta nova manera. També hem fet algunes modificacions addicionals a ShadeWidget, que comentarem de seguida. Al diagrama de classes només hi apareixen els mètodes més significatius de cada classe, que són els que comentarem aquí.

Les classes segueixen un esquema jeràrquic, amb HoverPoints al nivell més baix, ShadeWidget al nivell del mig i GradientEditor al nivell més alt.

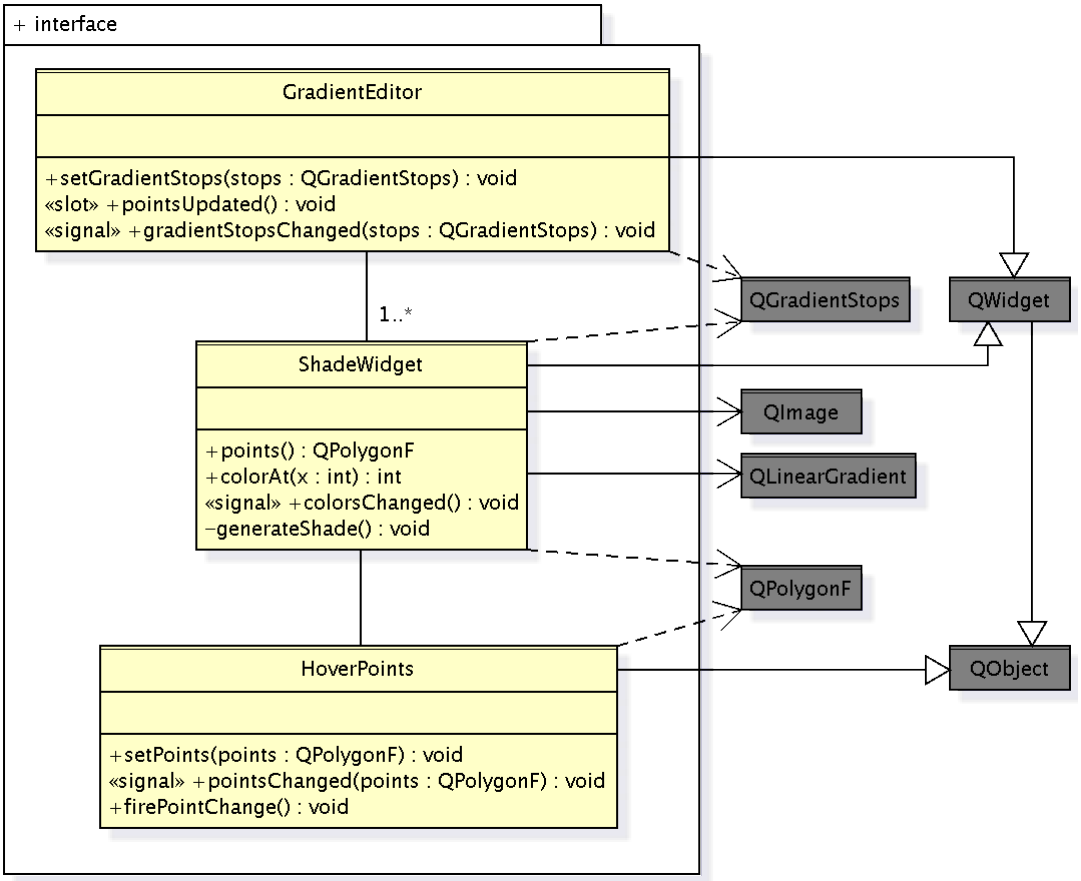


Figura 5.27: Diagrama de classes del submòdul de definició de funcions de transferència de les classes compartides.

HoverPoints és la classe que guarda els punts de la funció de transferència d'un component del color RGBA. S'encarrega de crear-los, esborrar-los, moure'ls, mantenir-los ordenats, etc. També és l'encarregada de dibuixar els punts sobre ShadeWidget. Se li poden assignar uns punts concrets amb setPoints(const QPolygonF&). Quan canvien els punts per la interacció de l'usuari emet el *signal* pointsChanged(const QPolygonF&) des del mètode firePointChange().

ShadeWidget és el *widget* on es veu dibuixada la funció de transferència d'un component del color RGBA. S'encarrega de guardar l'objecte HoverPoints i de dibuixar el fons sobre el qual es dibuixen els punts. Com a mètodes importants té:

- points(): retorna els punts del HoverPoints que guarda, emmagatzemats en un QPolygonF
- colorAt(int): retorna la intensitat del component en el punt passat com a paràmetre; si cal fa una interpolació per calcular-la
- generateShade(): aquest mètode dibuixa el fons del *widget*, que serà un degradat; té algunes modificacions per solucionar un problema amb les intensitats límit; el problema era que un punt no podia mai agafar un valor d'intensitat 0

També té el *signal* colorsChanged(), que s'emet quan el HoverPoints emet el seu *signal* pointsChanged(...).

Finalment, GradientEditor és el *widget* més important dels tres. Guarda 4 ShadeWidgets, un per cada component del color RGBA i és el que té la capacitat de retornar la funció de transferència completa.

Té el mètode setGradientStops(const QGradientStops&), que li assigna una funció de transferència completa, i s'encarrega de convertir-la en 4 funcions de transferència, una per cada component i assignar-les als ShadeWidgets corresponents, perquè cadascun dibuixi la seva part.

Hi ha l'*slot* pointsUpdated(), connectat al *signal* colorsChanged() de cada ShadeWidget, que fusiona les 4 funcions de transferència dels ShadeWidgets en una de sola, i llavors emet el *signal* gradientStopsChanged(const QGradientStops&) passant com a paràmetre aquesta funció de transferència completa.

QGradientStops representa una funció de transferència en l'espai [0,1] × RGBA. Concretament és un QVector<QGradientStop>, i QGradientStop és un QPair<qreal, QColor>.

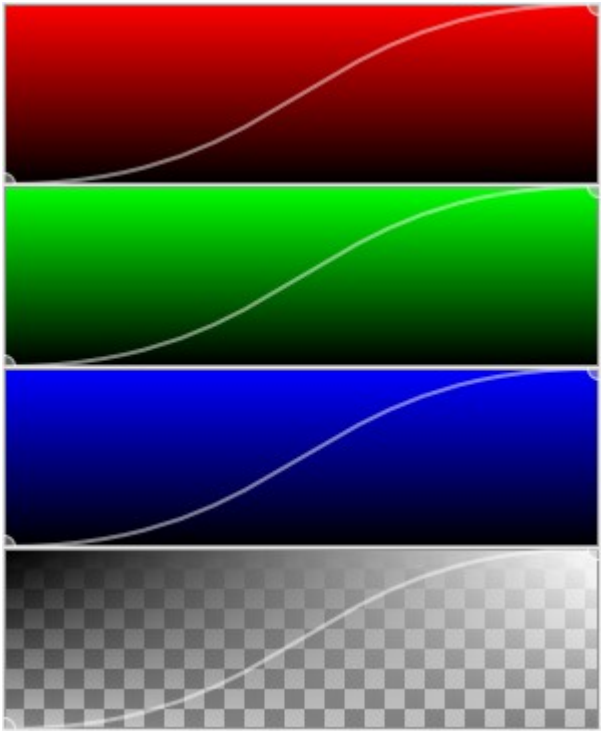


Figura 5.28: GradientEditor. Cadascun dels 4 rectangles és un ShadeWidget.

5.4.2 Disseny del submòdul de selecció de volums

El diagrama de classes específic d'aquest submòdul és el de la Figura 5.29.

Recordem que aquest submòdul inclou les classes que permeten seleccionar un o més volums d'entre els que estan oberts. Conté les classes **SelectVolumesDialogBase**, **SelectSingleVolumeDialog** i **SelectMultipleVolumesDialog**. Aquestes classes són molt senzilles.

SelectVolumesDialogBase només defineix la interfície comuna als dos tipus de selecció. És un quadre de diàleg modal amb una llista i un botó d'acceptar i un de cancel·lar.

Les dues classes filles consulten el nombre de volums a VolumeRepository i afegeixen un ítem a la llista per cada volum, amb el text "Volume id", on id és l'identificador del volum. Actualment no és possible consultar el nom d'un volum.

SelectSingleVolumeDialog fa que la llista sigui de selecció única, és a dir, que es pugui seleccionar només un element.

SelectMultipleVolumesDialog fa que la llista sigui de selecció múltiple, és a dir, que es pugui seleccionar qualsevol combinació d'elements.

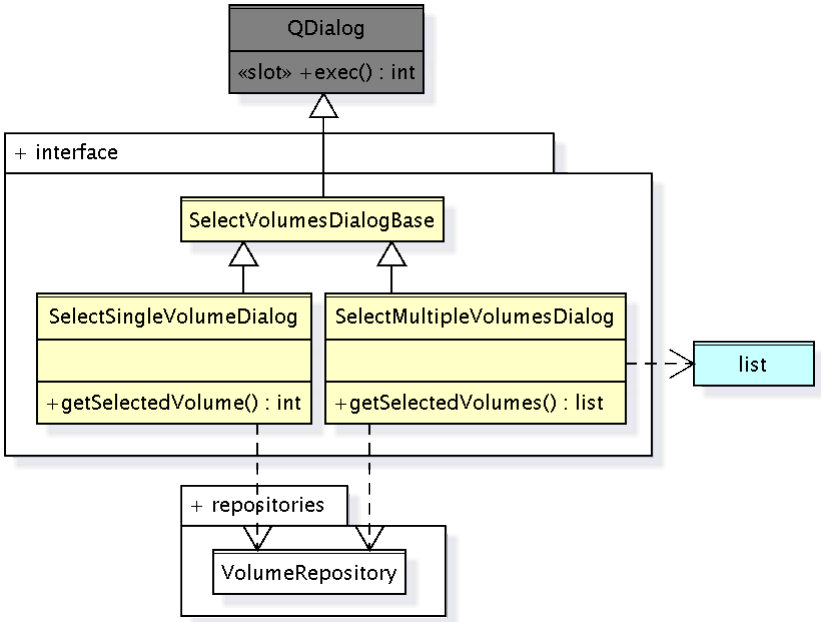


Figura 5.29: Diagrama de classes del submòdul de selecció de volums de les classes compartides.

Tots dos es mostren mitjançant el mètode exec() heretat de QDialog, i després es pot consultar el resultat de la selecció.

A SelectSingleVolumeDialog el mètode es diu getSelectedVolume() i retorna un enter corresponent a l'identificador del volum seleccionat.

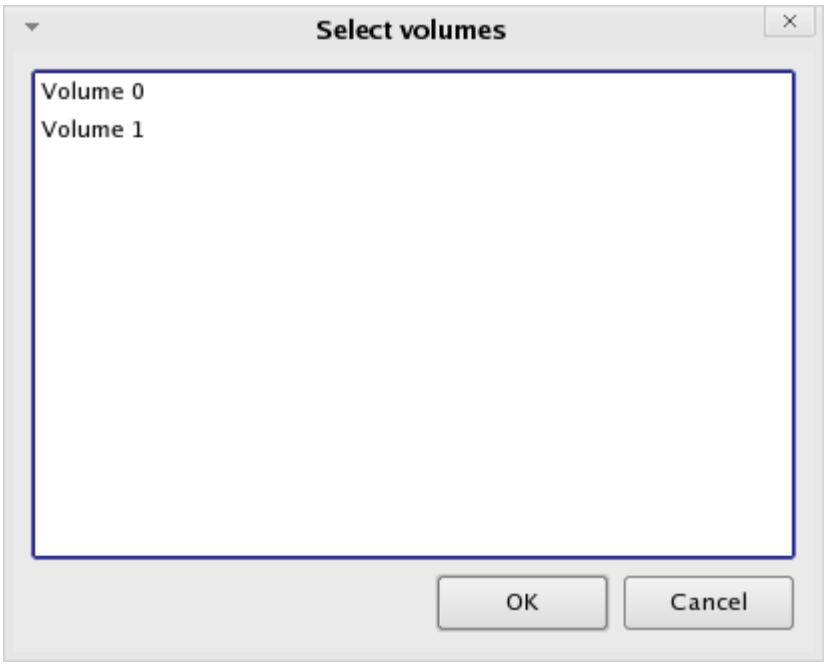


Figura 5.30: SelectSingleVolumeDialog o SelectMultipleVolumesDialog.

A SelectMultipleVolumesDialog el mètode es diu getSelectedVolumes() i retorna una llista d'enters corresponents als identificadors dels volums seleccionats.

5.4.3 Disseny del submòdul d'interacció amb la visualització

El diagrama de classes específic d'aquest submòdul és el de la Figura 5.31.

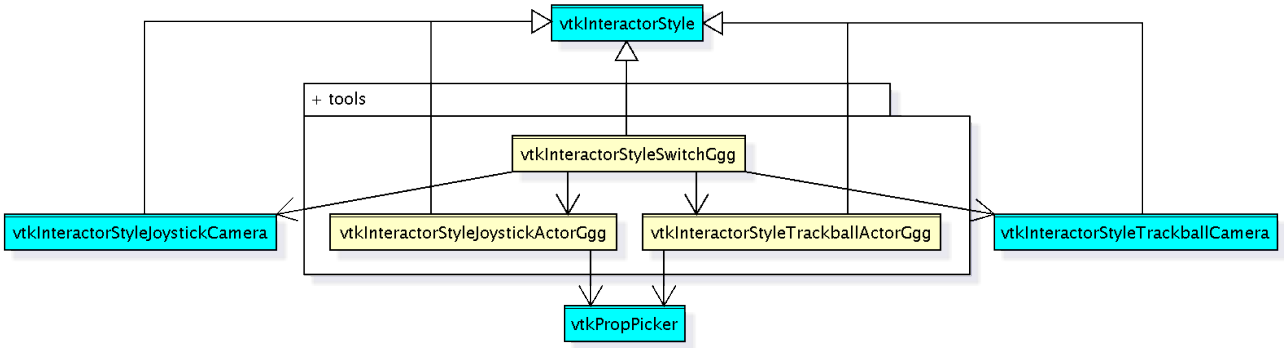


Figura 5.31: Diagrama de classes del submòdul d'interacció amb la visualització de les classes compartides.

Com ja hem dit, aquest submòdul està format per un conjunt de classes que permeten interaccionar amb les finestrs de visualització principals dels dos mètodes. En total són tres classes, i són les següents: **vtkInteractorStyleJoystickActorGgg**, **vtkInteractorStyleTrackballActorGgg** i **vtkInteractorStyleSwitchGgg**.

Aquestes tres classes estan basades en classes de VTK amb el mateix nom però sense el sufix Ggg.

vtkInteractorStyleJoystickActorGgg i vtkInteractorStyleTrackballActorGgg són versions modificades de les originals que fan servir un vtkPropPicker en lloc d'un vtkCellPicker per trobar l'actor seleccionat (el que està sota el ratolí). Amb això s'aconsegueix un guany important en el temps de resposta quan l'usuari vol girar o moure el model. El motiu és que vtkPropPicker fa servir el maquinari o sistema de visualització gràfic per triar l'actor ràpidament; en el cas d'un model de vòxels es fa servir la seva caixa envoltant per determinar si hi ha intersecció o no. Contràriament, vtkCellPicker fa servir un *ray casting* per determinar la cel·la (o vòxel) exacta intersecada, i aquest és un sistema molt més lent; a més a més no ens cal saber la cel·la exacta per moure el volum.

vtkInteractorStyleSwitchGgg és un versió modificada de l'original per fer servir les dues classes anteriors en lloc de les seves originals. Pels *interactors* de càmera es mantenen els originals.

6 Implementació

En aquest capítol explicarem el funcionament de l'aplicació a través de les diferents interfícies, i els efectes de cada paràmetre sobre la visualització.

6.1 StarViewer bàsic

Quan iniciem l'StarViewer el primer que veiem és una finestra com la següent:

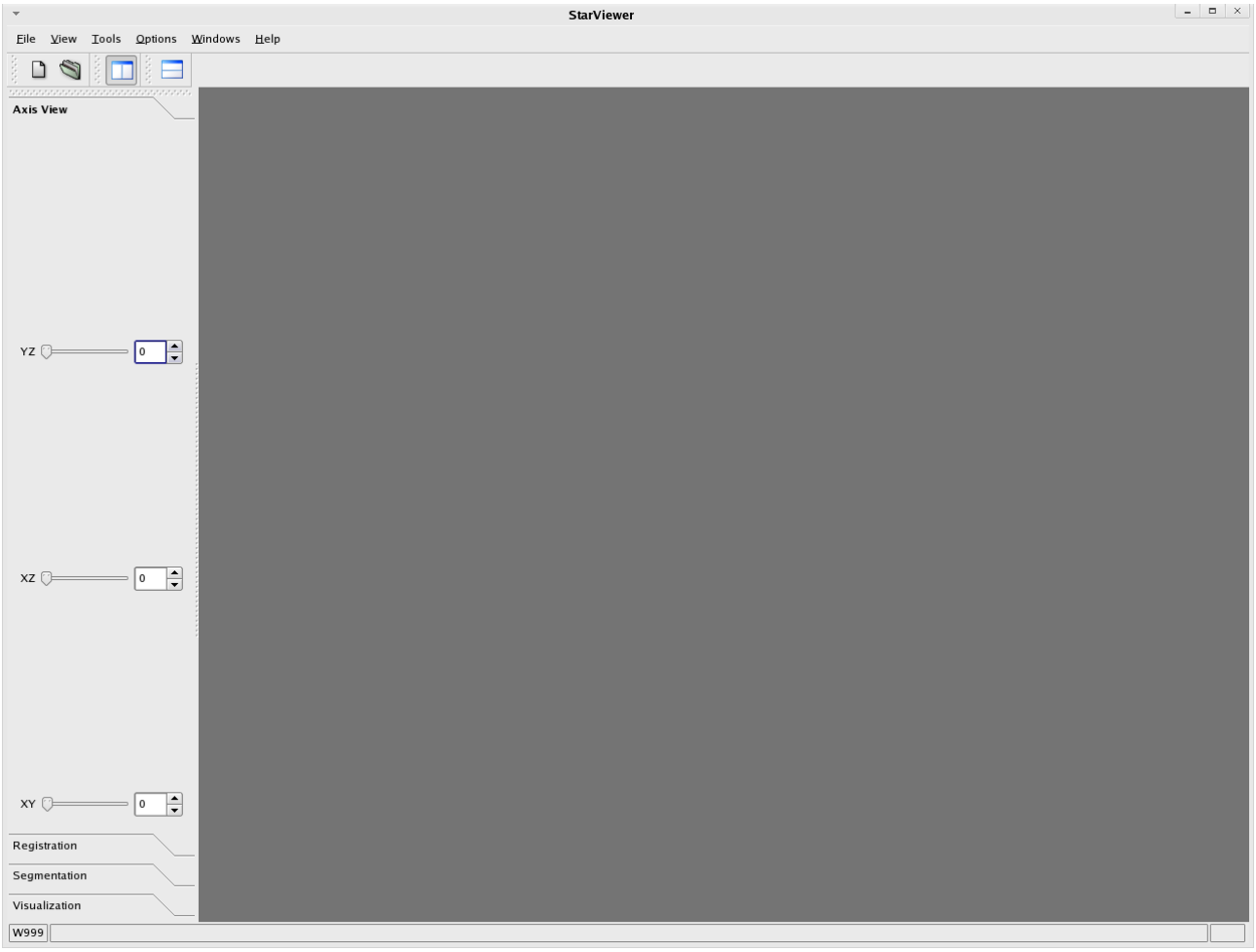


Figura 6.1: StarViewer.

En aquesta interfície hi ha diversos menús i botons, però el que ens interessa per ara és obrir un model. Això es pot fer de diverses maneres, per exemple fent clic al botó d'obrir, el que té una icona d'una carpeta. Al quadre de diàleg que apareix (Figura 6.2) només cal buscar el model i obrir-lo.

Per exemple, obrim el model "t1_n3_rf0.mhd". Llavors apareixeran les imatges de les llesques del model als quadrats de l'esquerra de la plataforma, com es veu a la Figura 6.3. Podem navegar per les llesques de qualsevol eix mitjançant aquesta eina.

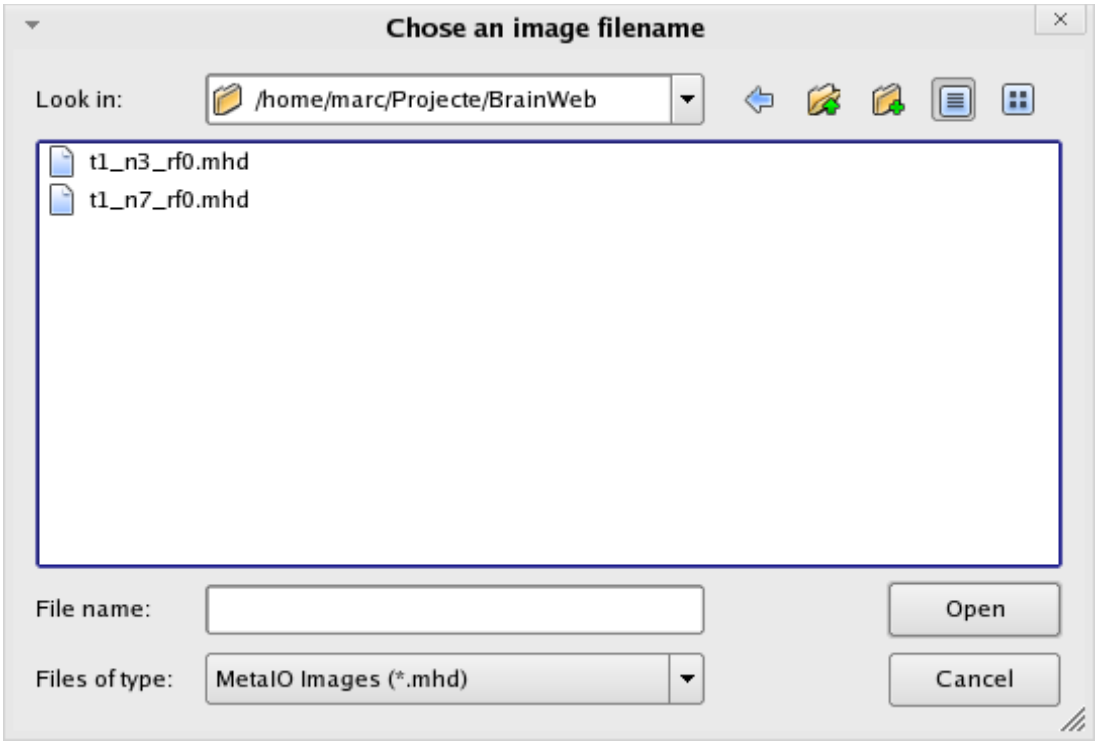


Figura 6.2: Quadre de diàleg per obrir un model.

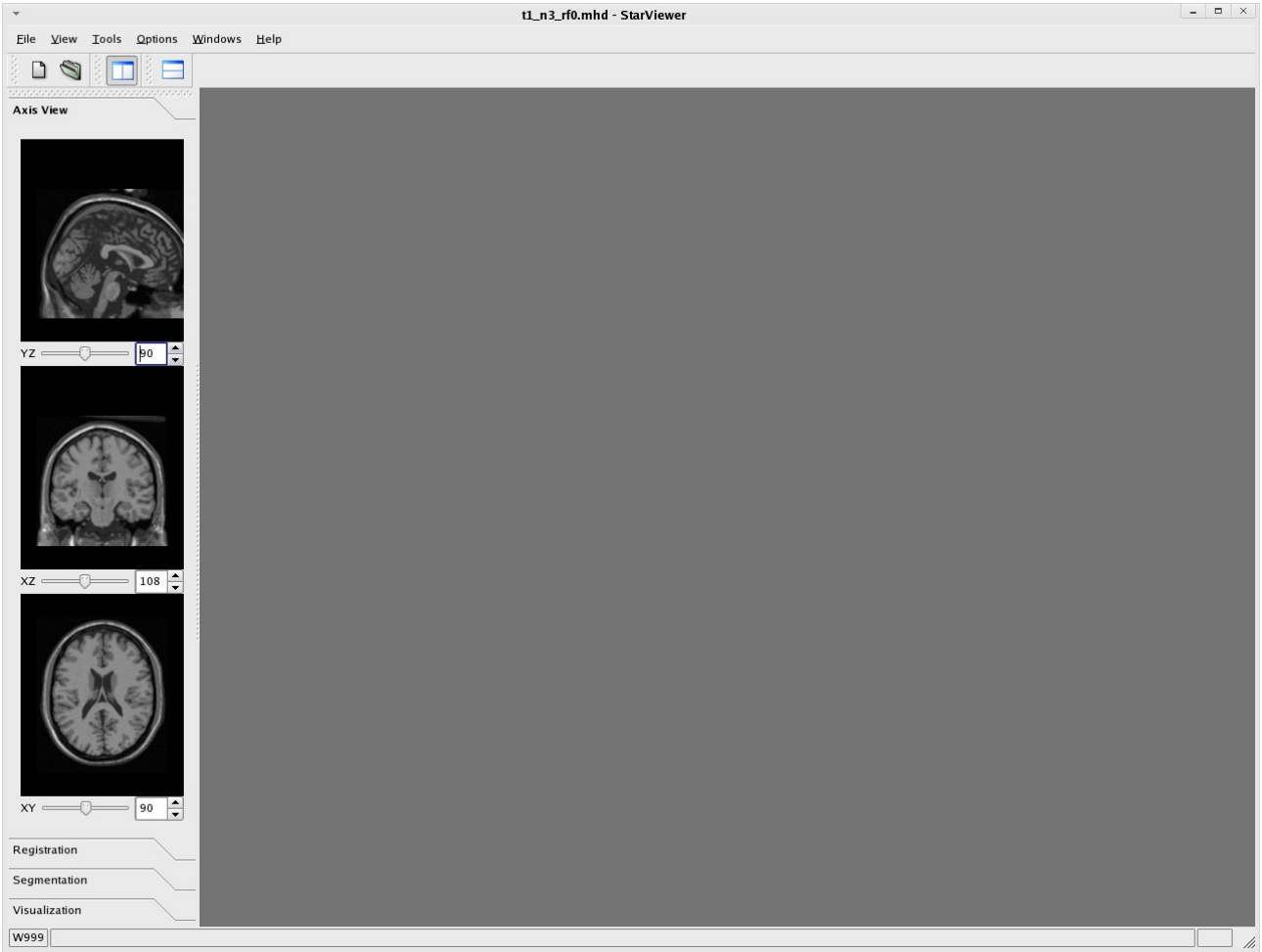


Figura 6.3: StarViewer després d'obrir un model.

Llavors podem accedir als diferents tipus de mètodes fent clic a la secció corresponent de la caixa d'eines. Per exemple a la secció "Visualization" podrem triar entre els dos mètodes de visualització implementats en aquest projecte. Tots dos apareixen llistats al quadre combinat. Per defecte està triada la primera opció, en aquest cas "Magic Mirrors". Això ho podem veure a la Figura 6.4.

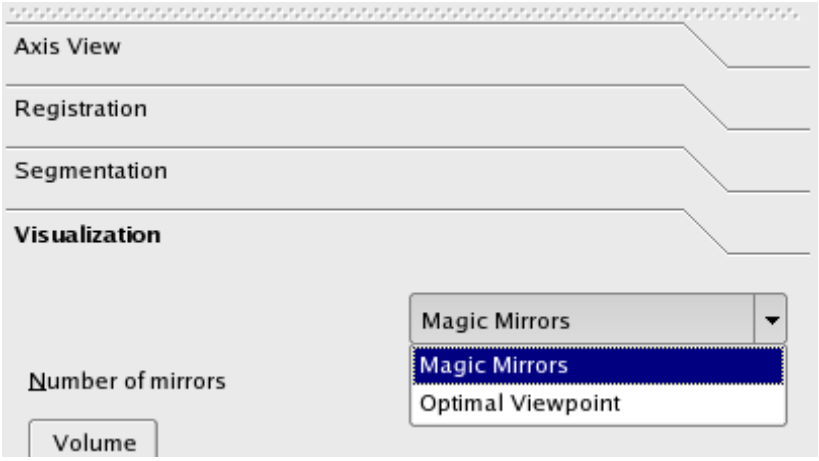


Figura 6.4: Mètodes de visualització.

6.2 Magic Mirrors

Veiem ara el funcionament del mètode dels Miralls Màgics. Primer veurem un exemple de visualització d'un model simple i després veurem les diferències que hi ha quan es tracta d'un model fusionat.

6.2.1 Models simples

Un cop hem obert un model i hem accedit a la part dels Miralls Màgics, el pas següent més lògic és seleccionar-lo per a la visualització. Això ho fem prement el botó "Select..." i triant el volum al quadre de diàleg que apareix (Figura 6.5). No obstant, també podríem començar creant els miralls i configurant-los.

Un cop seleccionat el volum apareixeran les opcions per configurar la visualització d'aquest (Figura 6.6).

Després posem 4 miralls i configurem la seva posició perquè estiguin repartits per darrere del model, amb totes les combinacions de les latituds -30 i 30 i les longituds -150 i 150. Mantenim les funcions de transferència predeterminades. (Figura 6.7)

En prémer el botó "Apply" apareix la finestra de visualització dels Miralls Màgics, tal com es veu a la Figura 6.8.

Ara podem interaccionar amb aquesta visualització utilitzant el ratolí, i també el teclat. Amb la configuració inicial el ratolí mou la càmera, però prement la tecla 'A' podem moure el model, i prement la tecla 'C' tornem a controlar la càmera. Arrossegant el ratolí mantenint premut el botó

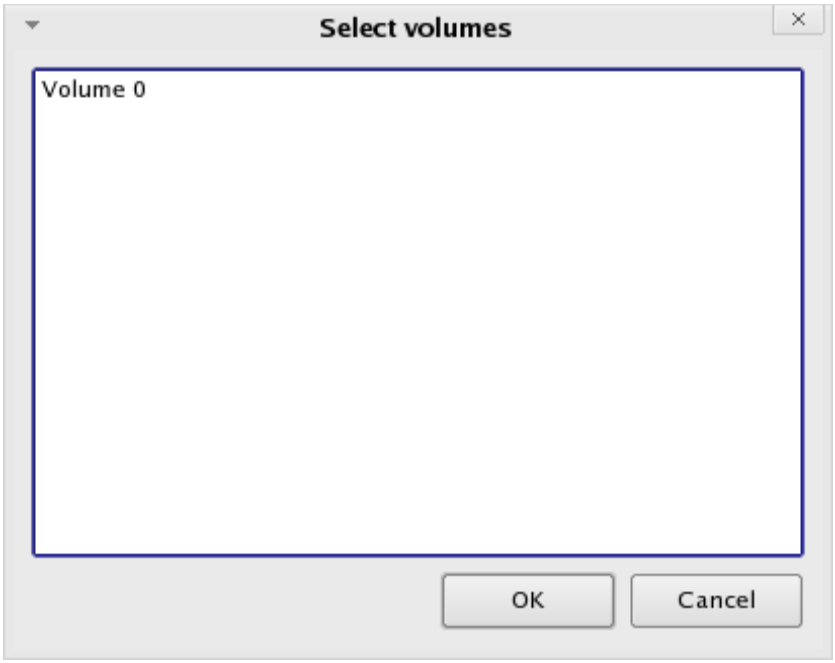


Figura 6.5: Quadre de diàleg de selecció de volums.

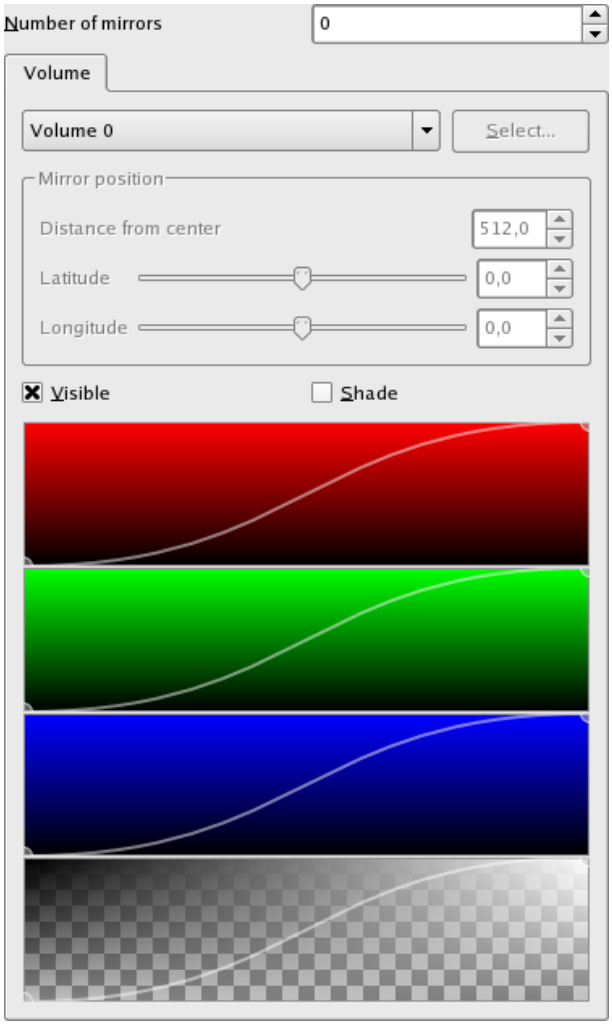


Figura 6.6: Opcions de configuració dels Miralls Màgics.

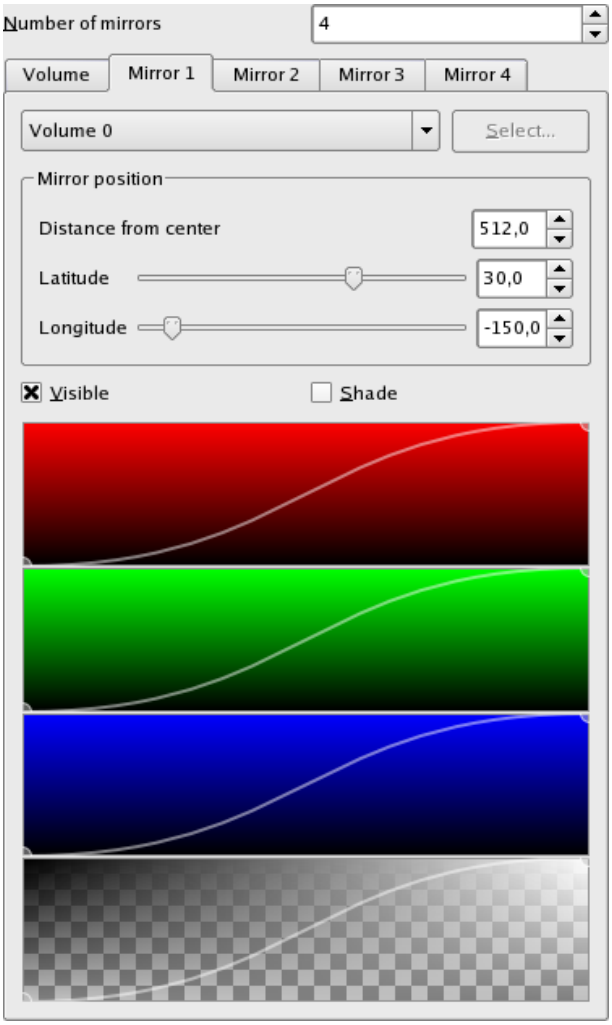


Figura 6.7: 4 miralls amb la posició configurada.

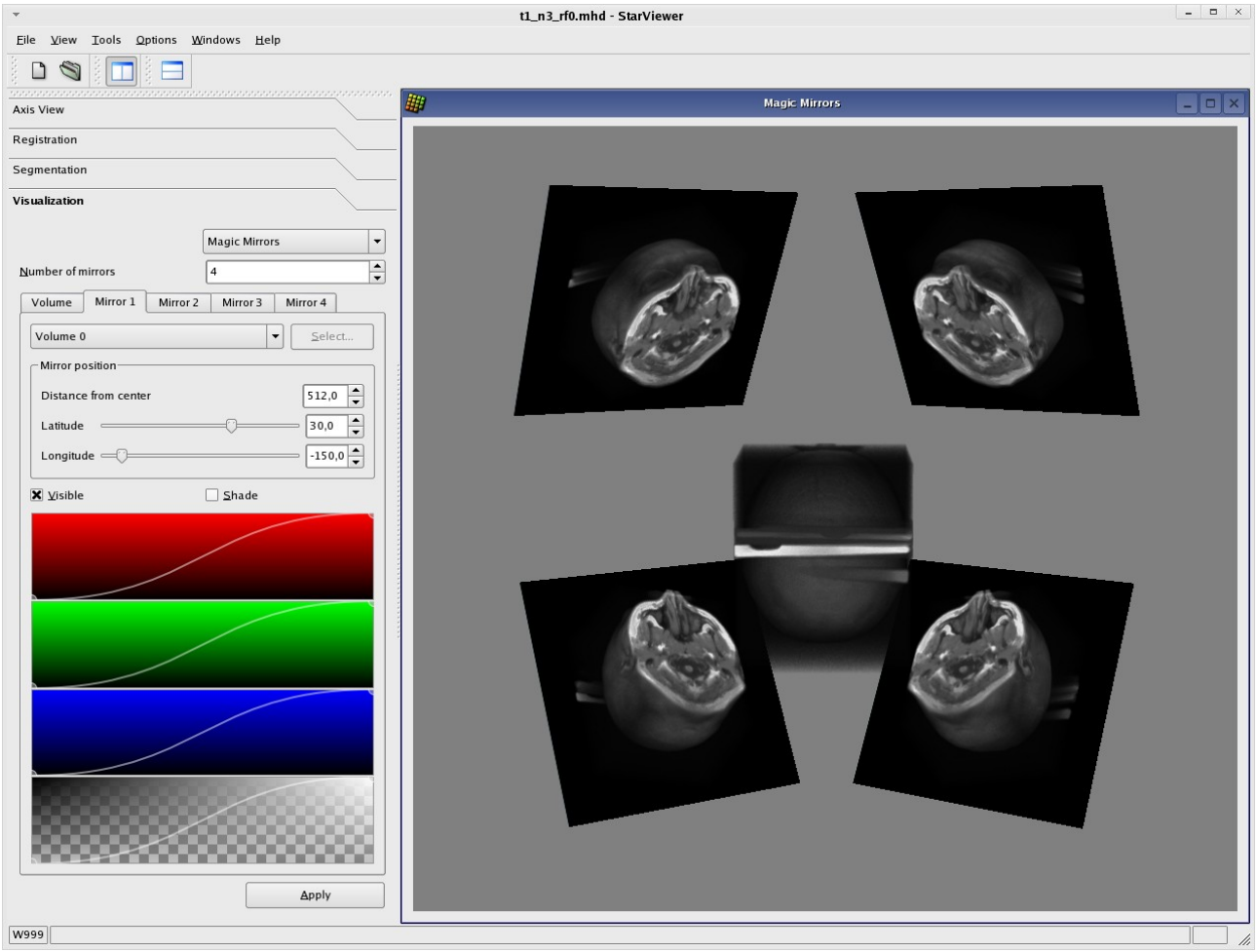


Figura 6.8: Miralls Màgics.

esquerre podem girar la càmera al voltant del model o girar el model. Si arrosseguem amb el botó del mig o la roda es desplaça la càmera o el model. Amb el botó dret es fa zoom o s'escala el model. Girant la rodeta també podem fer zoom si estem controlant la càmera, però no té cap efecte sobre el model.. A més a més hi ha dos estils d'interacció amb cada element: *joystick* i *trackball*. Al primer (predeterminat) s'hi accedeix prement la tecla 'J', i al segon amb la tecla 'T'. També podem prémer la tecla 'R' per posar la càmera a una distància que permeti veure el volum i tots els miralls.

A la Figura 6.10 podem hem mogut la càmera i a la Figura 6.12 hem girat el model.

La imatge dels miralls s'actualitza automàticament quan acaba la interacció amb el model. La interacció amb la càmera no afecta els miralls.

Ara comentarem els elements de la interfície per introduir els paràmetres d'entrada del mètode.

Primer de tot tenim un *spin box* que ens permet introduir el nombre de miralls que volem, amb un mínim de 0 i un màxim de 20. El nombre de miralls es pot canviar en qualsevol moment.

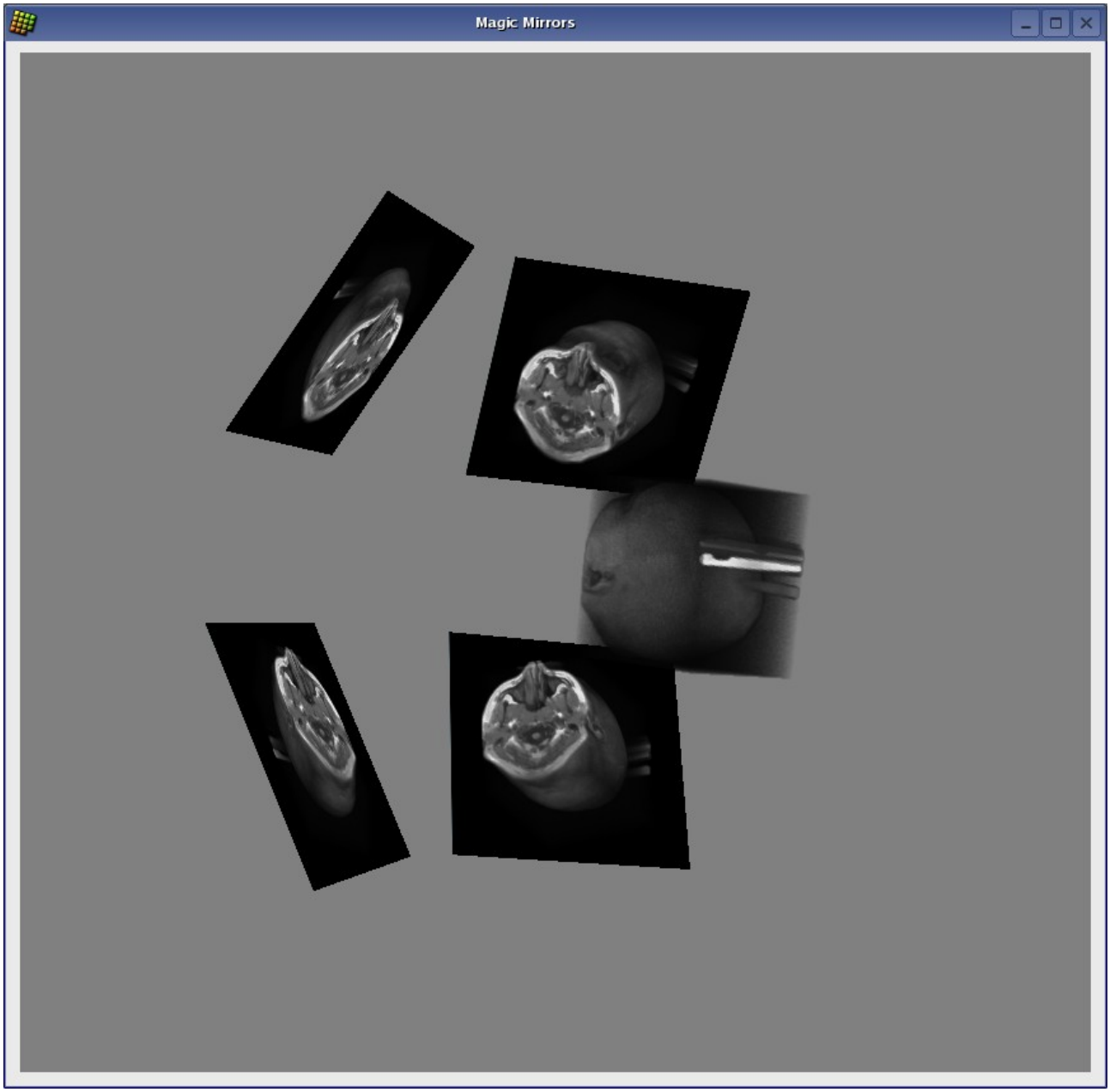


Figura 6.10: Miralls Màgics després d'haver girat la càmera.

Figura 6.9: Spin box *per introduir el nombre de miralls.*

Just a sota hi ha les pestanyes que permeten triar el mirall que volem configurar. Sempre hi ha una pestanya pel volum central i una per cada mirall.

Figura 6.11: Pestanyes de selecció de mirall.

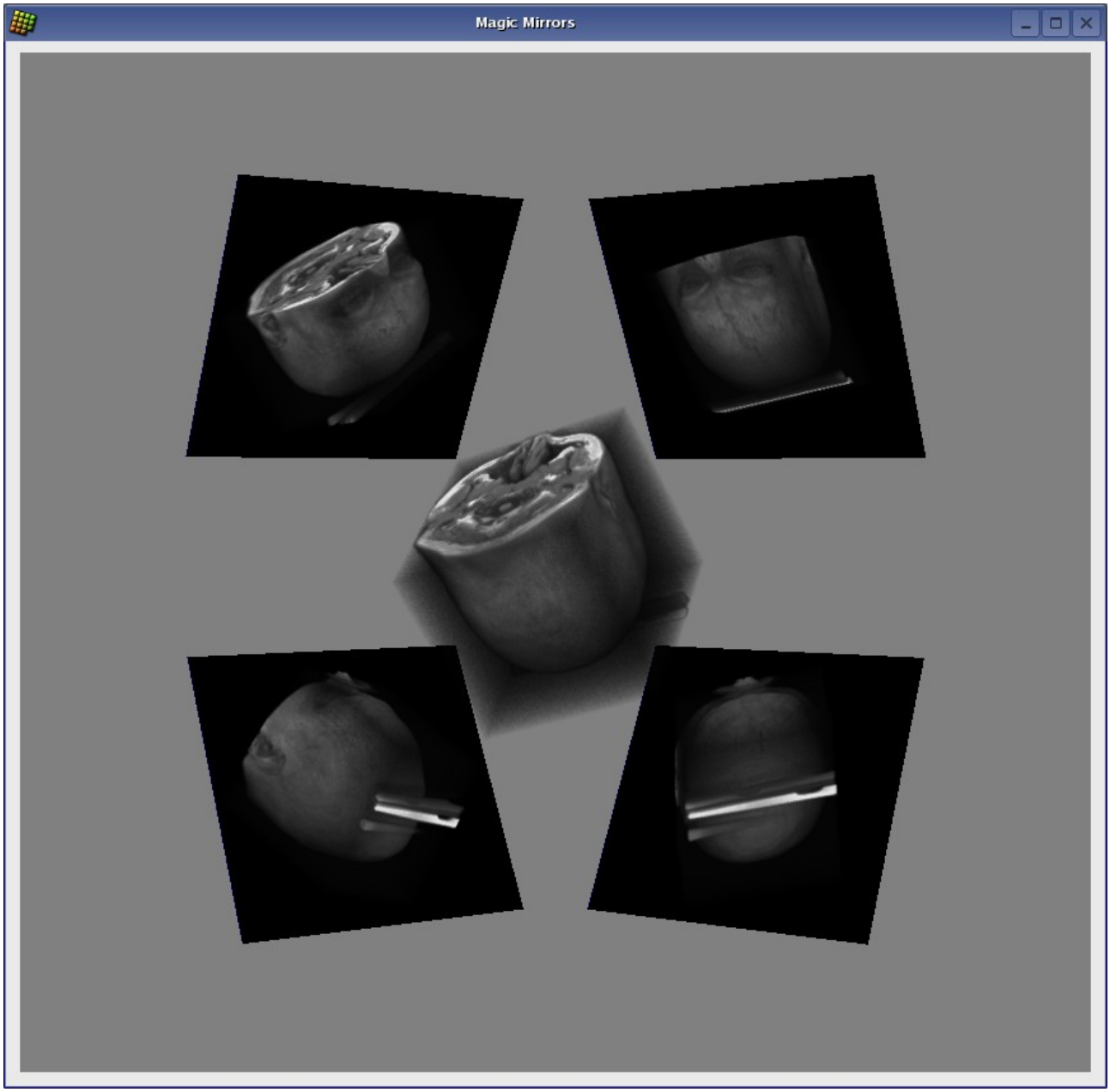


Figura 6.12: Miralls Màgics després d'haver girat el model.

Aquí s'acaben els elements "únics". Dels elements de la interfície següents n'hi ha un a cada pestanya.

El primer és un quadre combinat que permet seleccionar el volum o propietat que volem configurar dins d'aquest mirall.

Figura 6.13: Quadre combinat de selecció de volum.

Al seu costat hi ha el botó “Select...”, que mostra el diàleg de selecció de volums.

Figura 6.14: Botó per seleccionar els volums.

A sota hi ha el grup de controls de la posició del mirall (Figura 6.18). A la pestanya del volum estan desactivats ja que no tenen sentit. Aquests controls permeten configurar la distància del mirall al centre, la latitud del mirall i la longitud del mirall. La distància es pot introduir amb un *spin box*, i la latitud i la longitud amb un *slider* o un *spin box* indistintament. La latitud i la longitud són en graus i accepten valors reals en els rangs [-90, 90] i [-179.9, 180], respectivament. Podem veure els efectes de cada paràmetre a les imatges següents: a la Figura 6.15 hem apropat el mirall 1 fins a una distància de 256; a la Figura 6.16 hem canviat la latitud a 7.2; a la Figura 6.17 hem canviat la longitud a 19.2.

Aquí s’acaben els elements “de mirall”. Dels elements de la interfície següents n’hi ha un per cada volum a cada pestanya. Podem accedir als controls d’un volum concret mitjançant el quadre combinat comentat anteriorment.

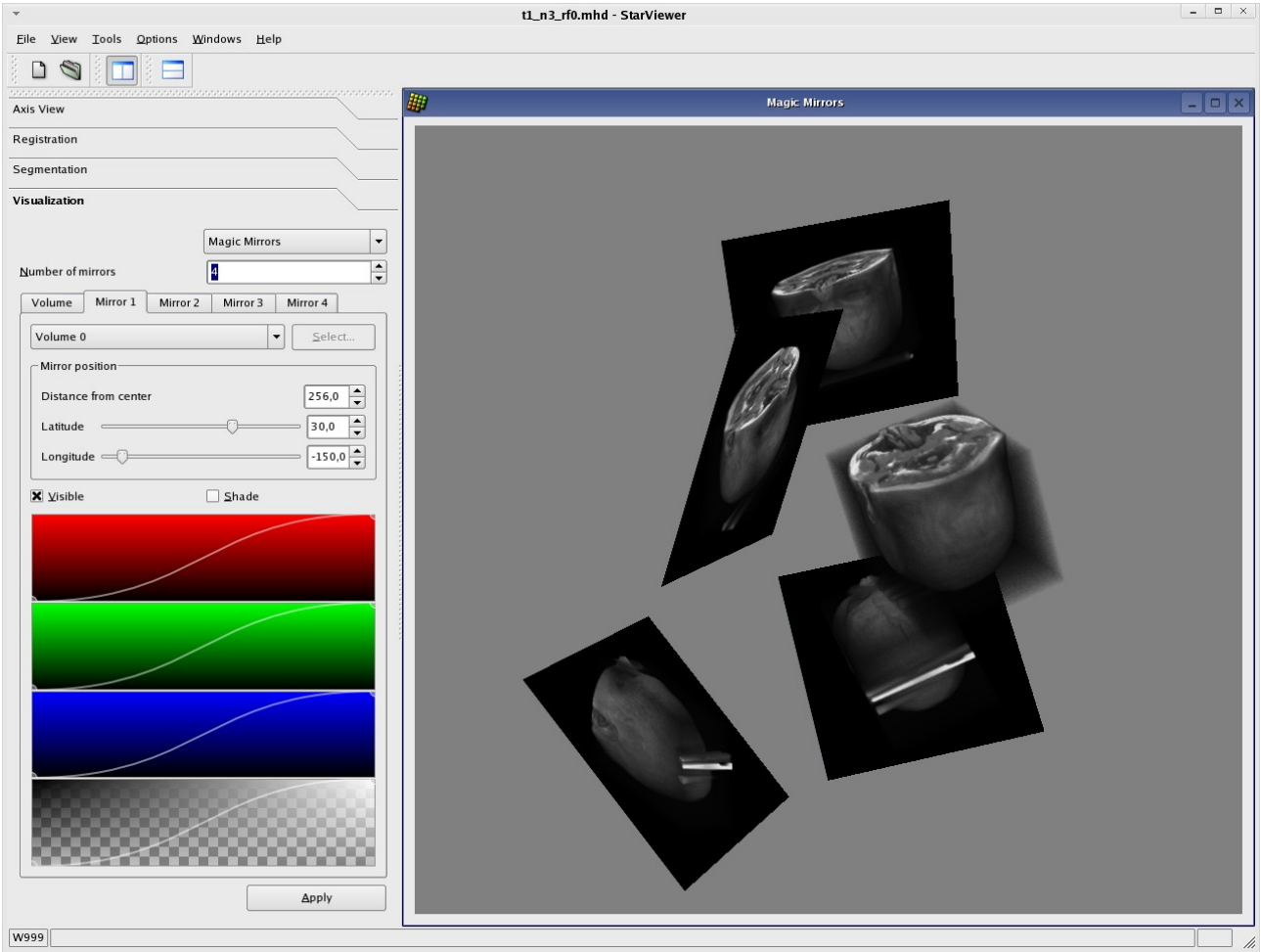


Figura 6.15: S’ha apropat el mirall 1.

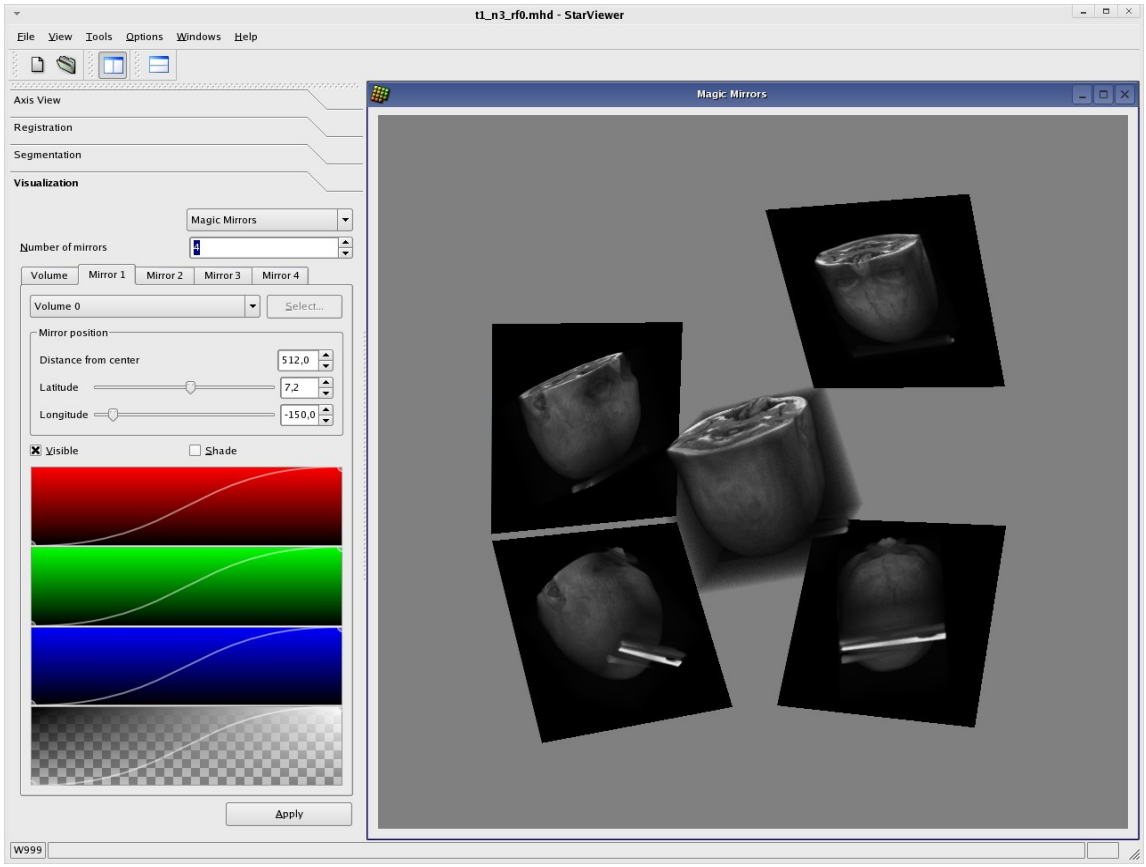


Figura 6.16: S'ha canviat la latitud del mirall 1.

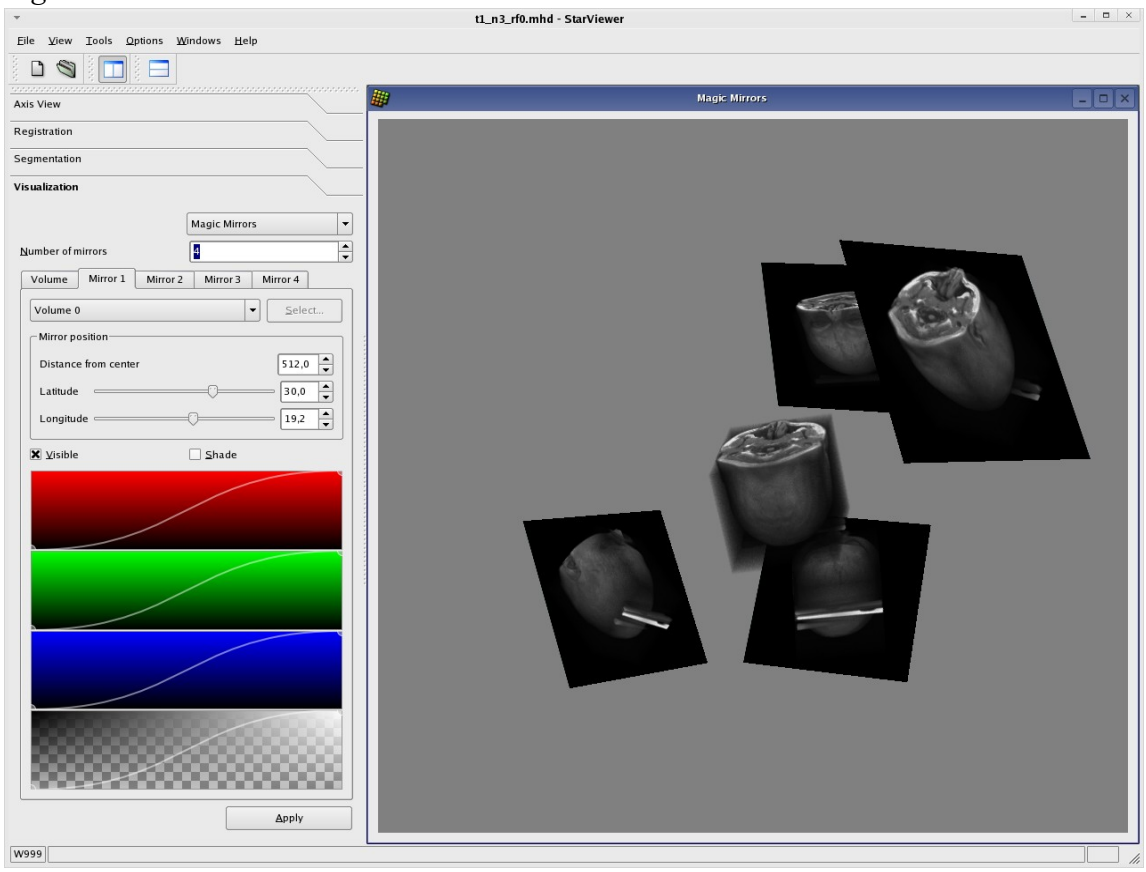


Figura 6.17: S'ha canviat la longitud del mirall 1.

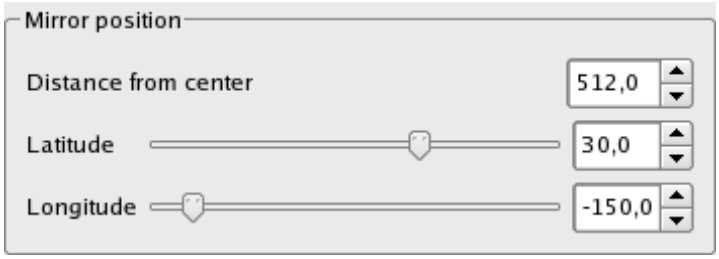


Figura 6.18: Grup de controls de la posició del mirall.

El primer d'aquests elements és el quadre de verificació "Visible". Sempre està marcat per defecte, i això fa que el volum es vegi. A la Figura 6.20 veiem l'efecte de desmarcar aquest quadre de verificació per al volum central. Aquesta opció té utilitat en la visualització de models fusionats.

Figura 6.19: Quadre de verificació "Visible".

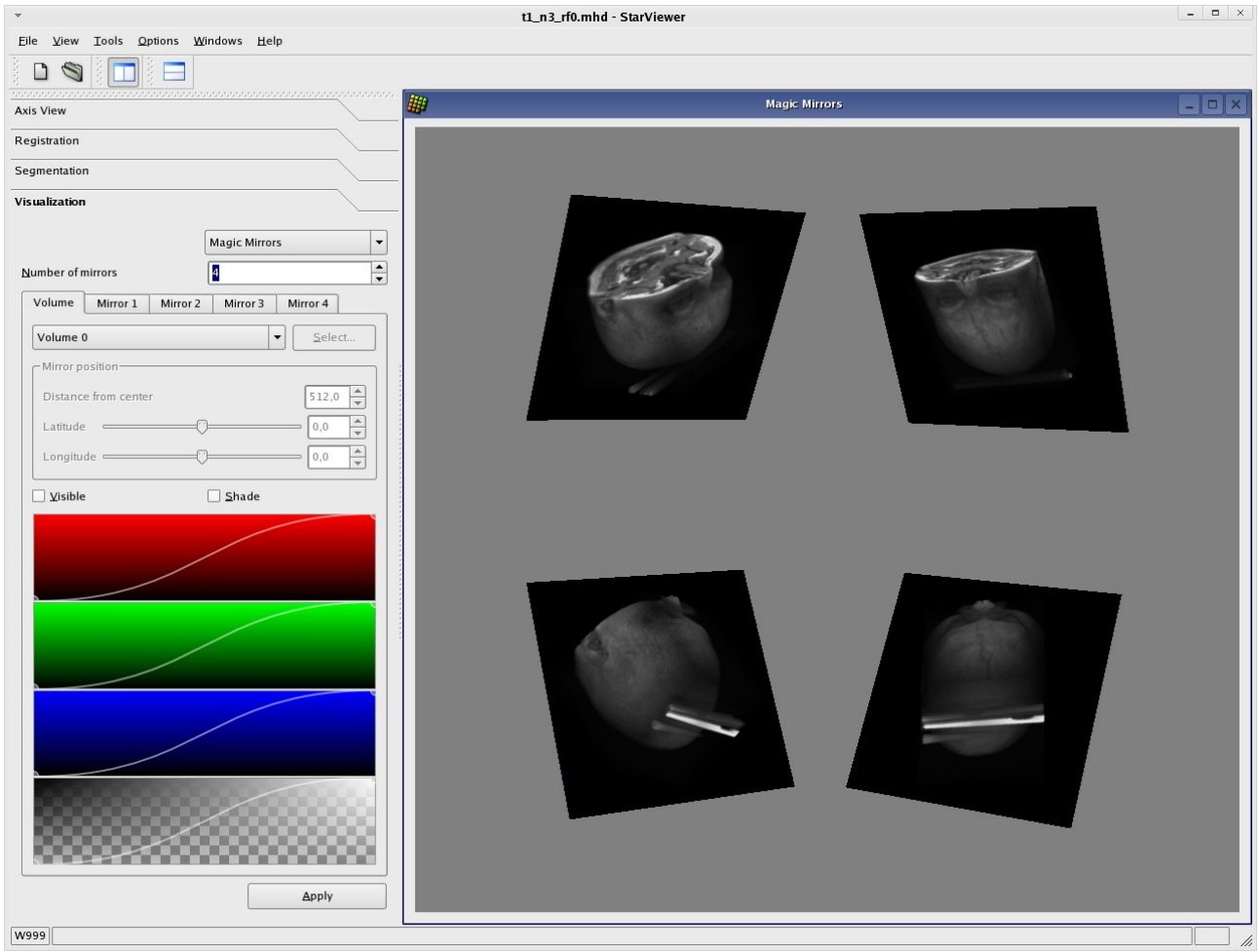


Figura 6.20: S'ha fet invisible el volum central.

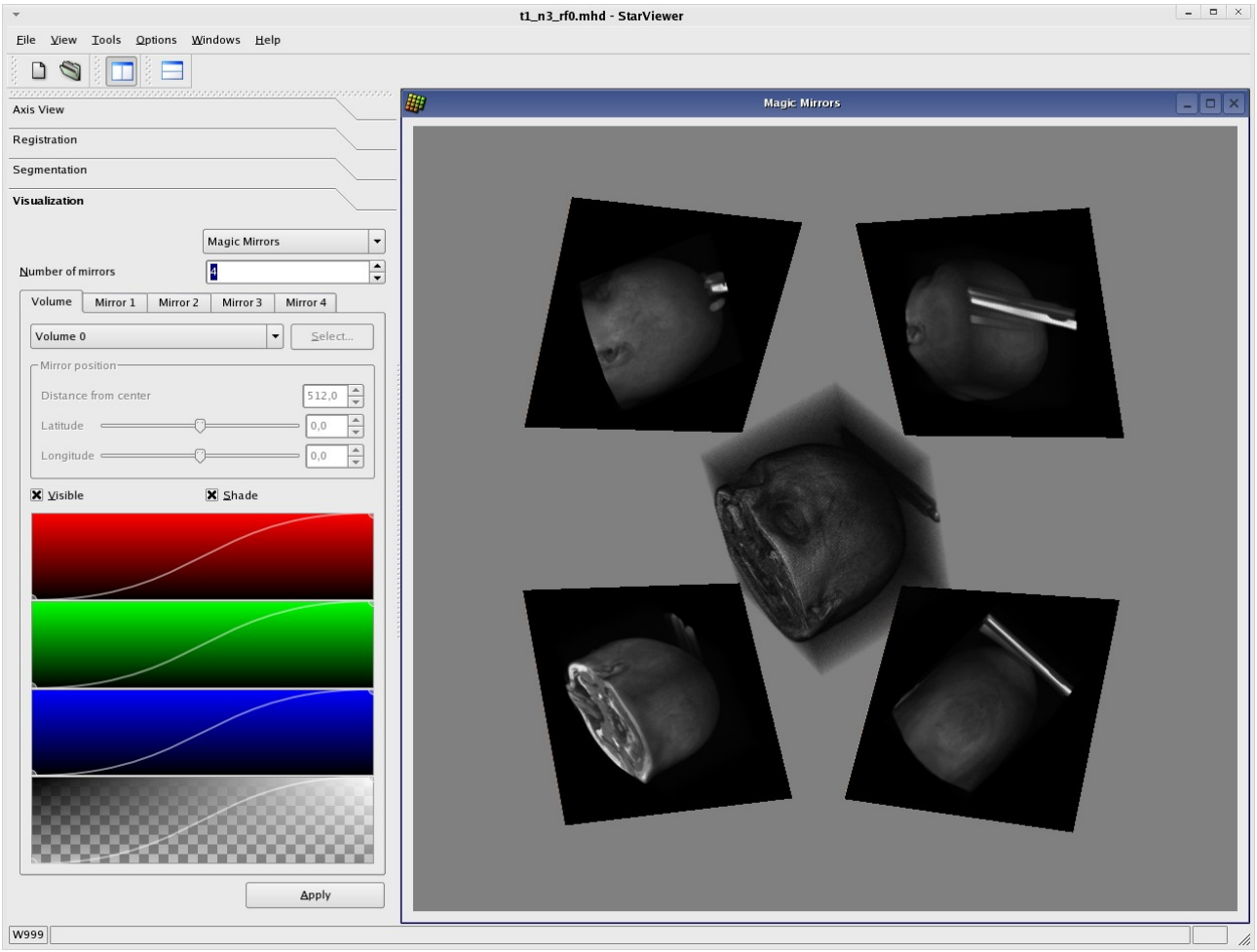


Figura 6.21: S'ha aplicat ombrejat al volum central.

A la seva dreta hi ha el quadre de verificació "Shade". Sempre està desmarcat per defecte i quan s'activa fa que s'apliqui un ombrejat al volum. A la Figura 6.21 veiem l'efecte de marcar aquesta casella per al volum central. Aquest efecte fa que en determinats casos la imatge es vegi millor, però la visualització és més lenta.

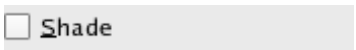


Figura 6.22: Quadre de verificació "Shade".

L'últim element de la interfície és l'editor de la funció de transferència. Podem editar cada component per separat. L'eix d'abscisses de cada funció representa els valors de propietat, i el d'ordenades la intensitat del component del color. Podem crear un nou punt fent clic amb el botó esquerre sobre un espai buit, esborrar-ne un fent clic sobre ell amb el botó dret i moure'n un arrossegant-lo amb el botó esquerre. A la Figura 6.24 veiem l'efecte d'aplicar una nova funció de transferència sobre el volum central.

Les imatges següents ens mostren diferents configuracions de visualització amb un model simple.

A la Figura 6.25 hem fet invisible el model al mirall 1, hem canviat la funció de transferència al mirall 3 i hem allunyat el mirall 4.

A la Figura 6.26 hem mogut més miralls.

A la Figura 6.27 hem tret un mirall i hem canviat funcions de transferència, a més de desactivar l'ombrejat al volum central.

A la Figura 6.28 col·loquem bé els tres miralls i posem 3 funcions de transferència més realistes. També hem activat l'ombrejat a tot arreu.

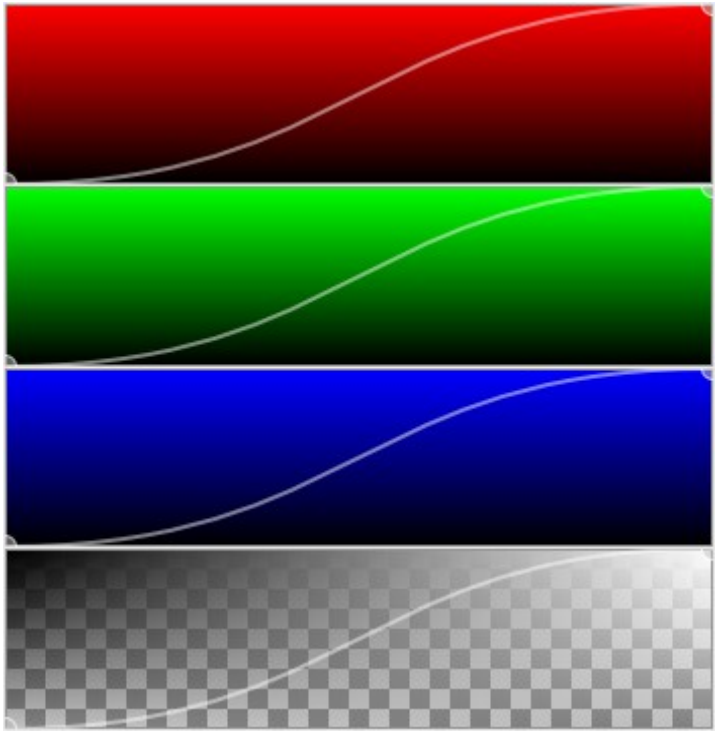


Figura 6.23: Editor de la funció de transferència.

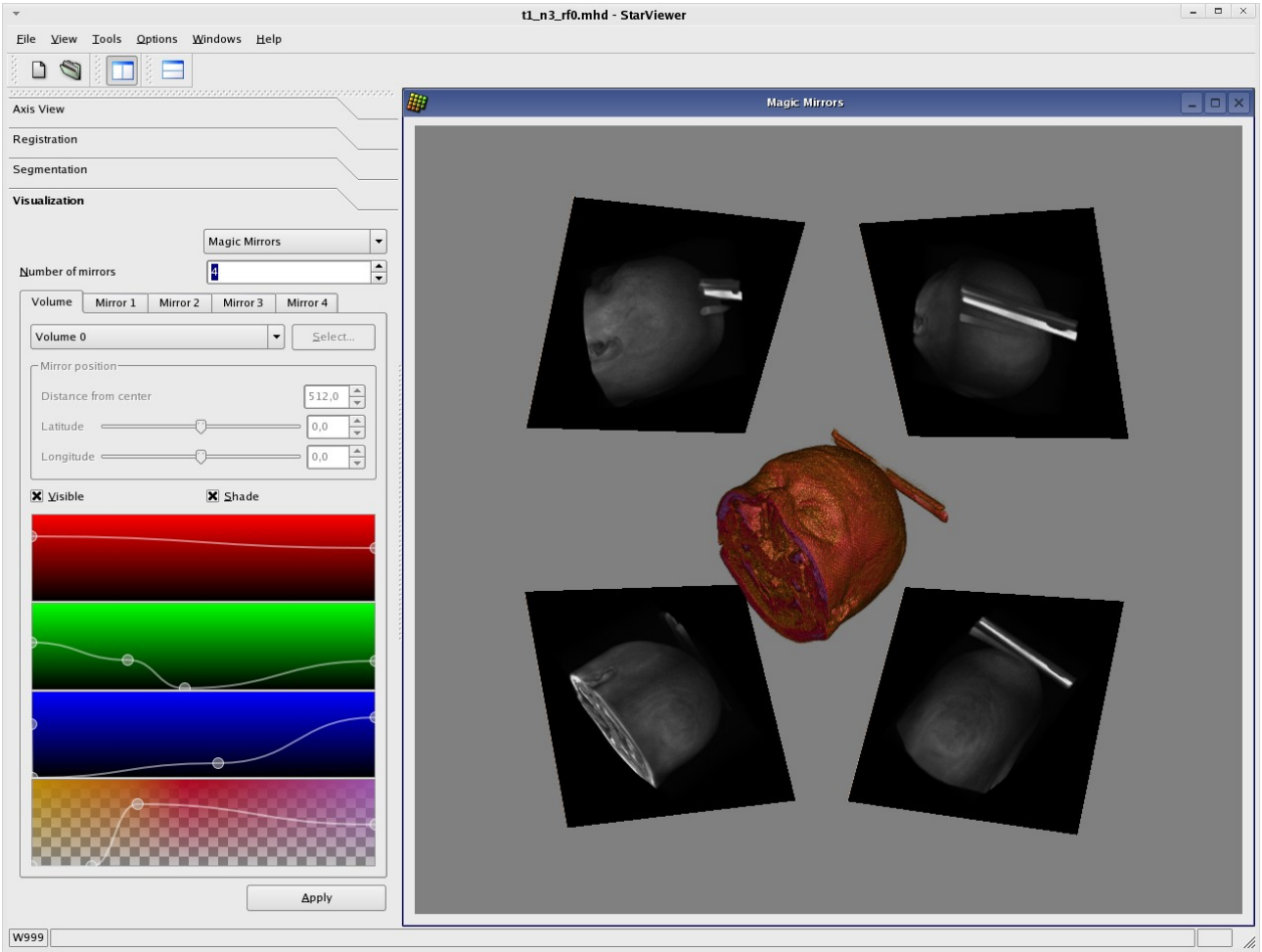


Figura 6.24: S'ha canviat la funció de transferència del volum central.

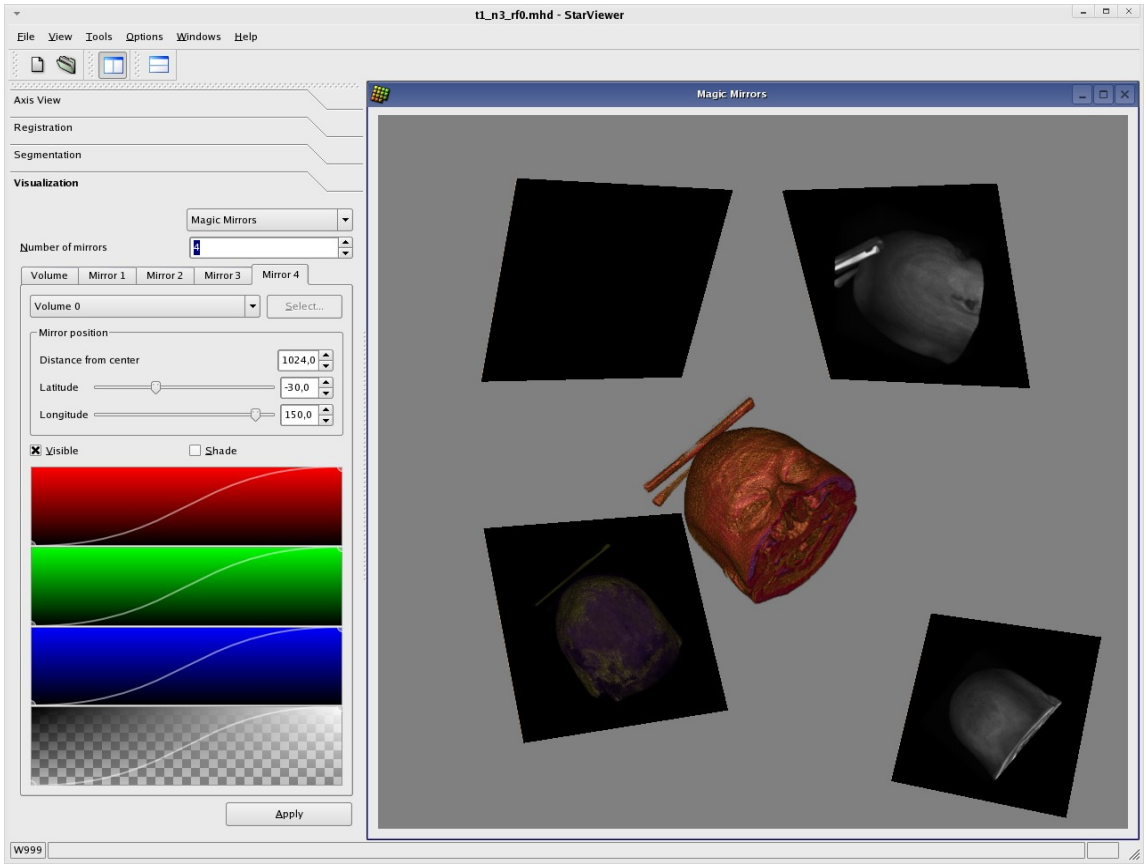


Figura 6.25: Miralls Màgics amb el model simple "t1_n3_rf0.mhd".

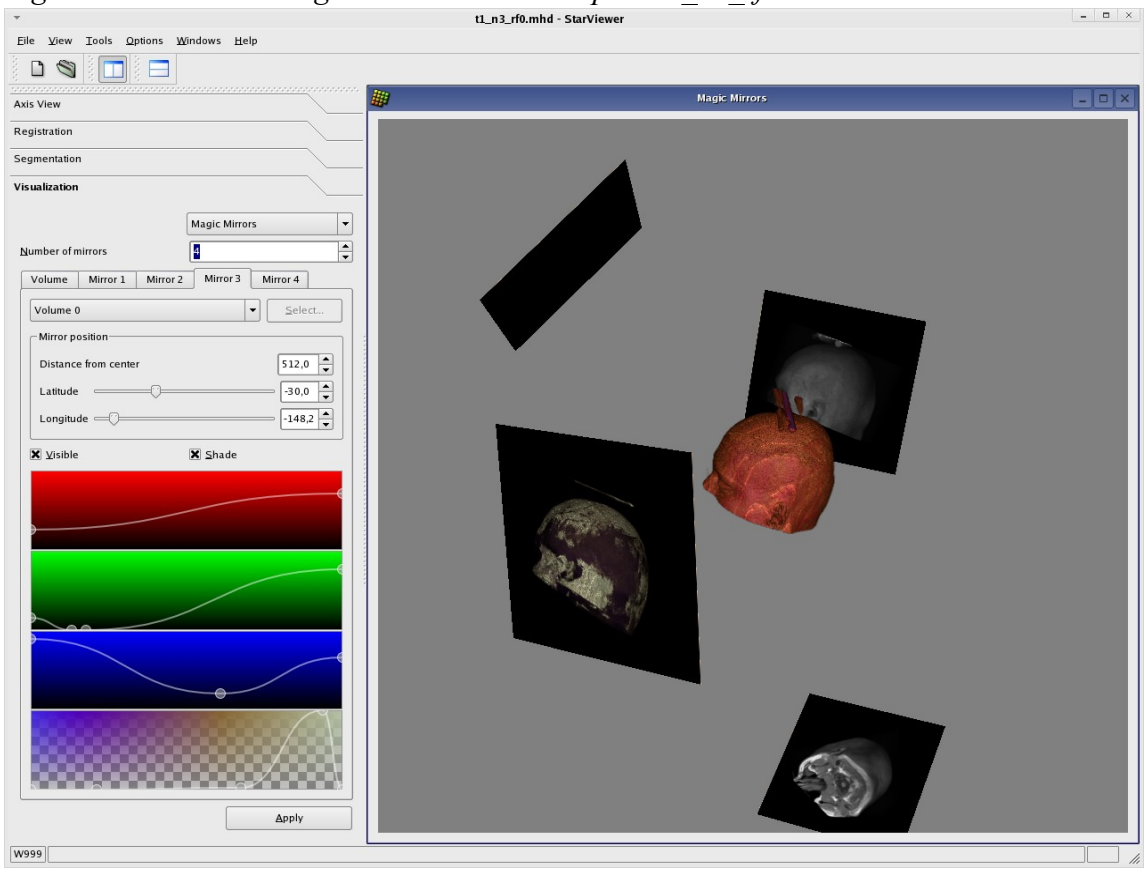


Figura 6.26: Miralls Màgics amb el model simple "t1_n3_rf0.mhd".

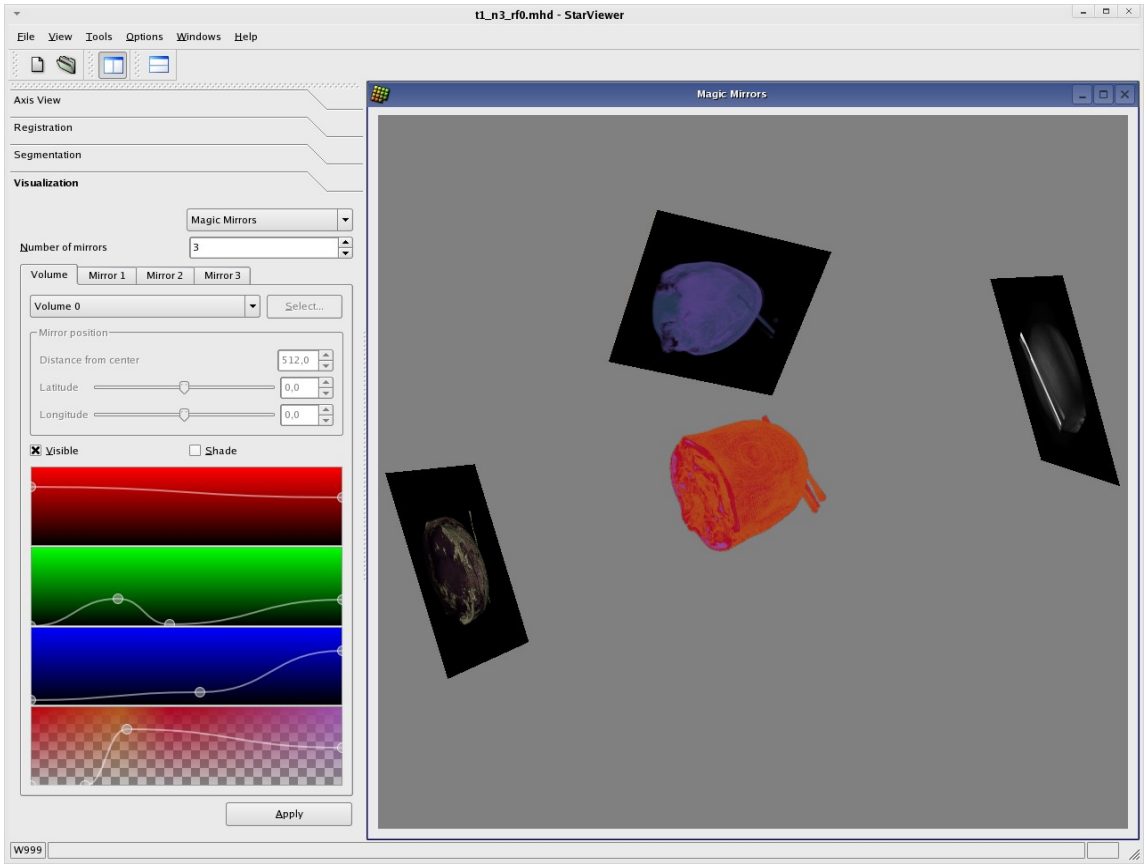


Figura 6.27: Miralls Màgics amb el model simple "t1_n3_rf0.mhd".

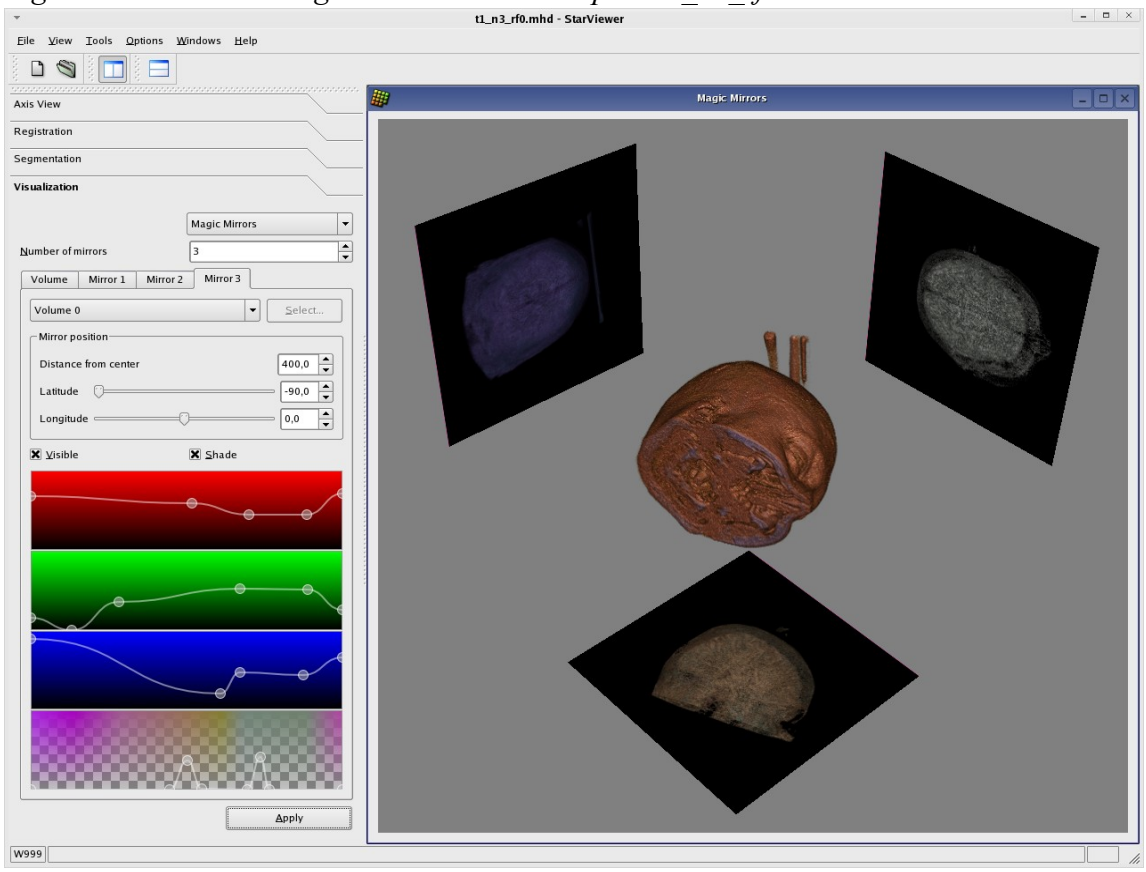


Figura 6.28: Miralls Màgics amb el model simple "t1_n3_rf0.mhd".

6.2.2 Models fusionats

De seguida veurem que treballar amb un model fusionat no és gaire diferent de treballar amb un model simple. Aquí veurem quines són les diferències més significatives entre els dos processos.

La diferència més important és que cal obrir dos models i aplicar-los un mètode de registre abans d'iniciar la visualització amb els Miralls Màgics.

La versió bàsica de l'StarViewer porta un mètode de registre com a exemple de com s'ha d'implementar un nou mètode a la plataforma. Aquest mètode treballa amb dues imatges obertes i mou la segona perquè s'alineï amb la primera, però no la mou directament sinó que en mou una còpia. Com que aquesta còpia registrada no es guarda al repositori no podem accedir-hi des del mètode dels Miralls Màgics, o sigui que per explicar els exemples treballarem amb dos models i suposarem que estan correctament registrats, encara que en realitat no ho estan.

Per explicar el funcionament dels Miralls Màgics amb models fusionats treballarem amb els models "pat1ct.mhd" i "pat1mrT1.mhd".

Un cop oberts i fusionats, el que hem de fer és seleccionar-los per a la visualització. Això ho podem fer mitjançant el botó "Select..." i seleccionant els dos volums que apareixeran llistats al quadre de diàleg.

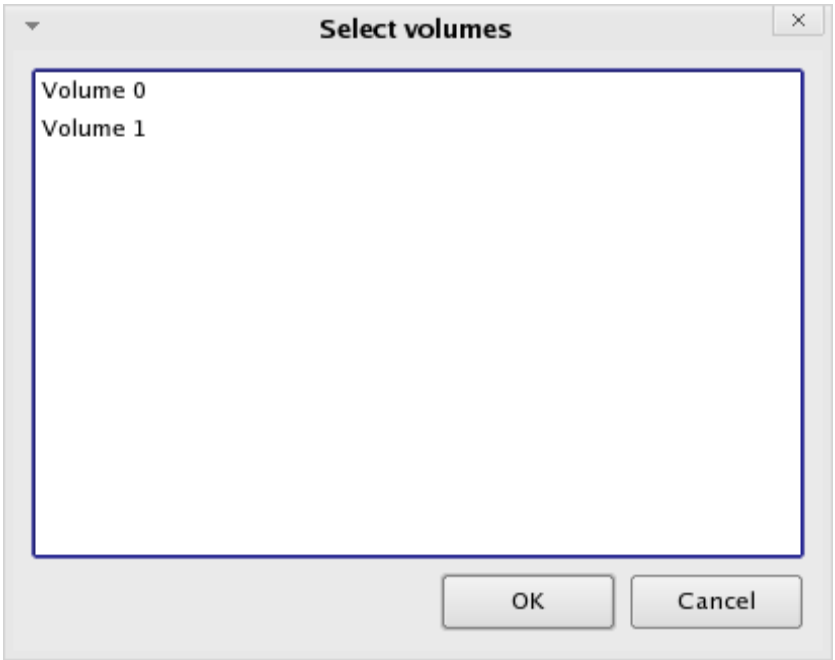


Figura 6.29: Quadre de diàleg de selecció de volums.

Llavors tenim les mateixes opcions que en el cas de models simples. Per exemple, creem 3 miralls i els col·loquem a les posicions[6] (0, -90), (0, 180) i (-90, 0). Amb això i girant una mica la càmera obtenim la imatge de la Figura 6.30.

6 La sintaxi de les posicions és (latitud, longitud).

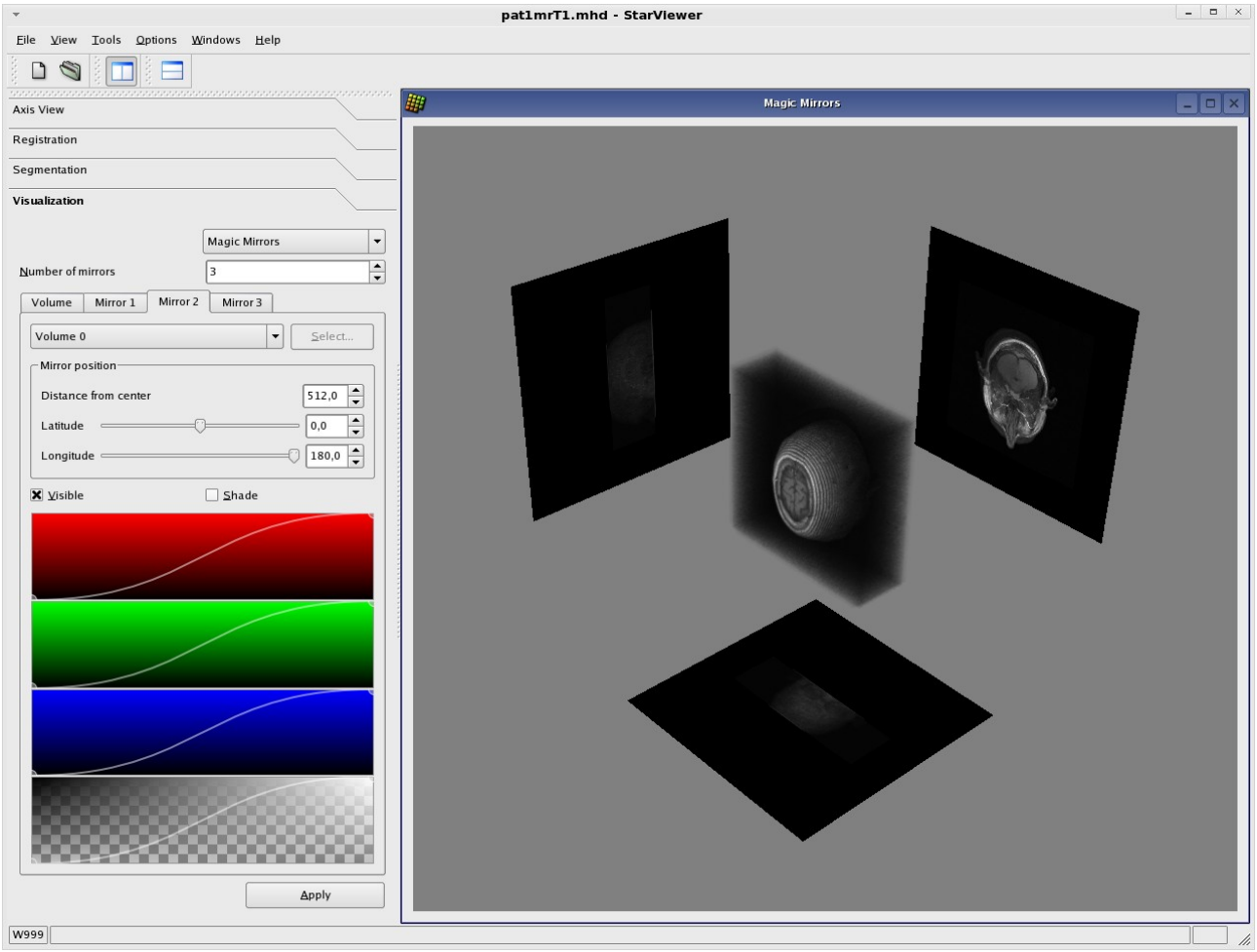


Figura 6.30: Miralls Màgics amb un model fusionat i 3 miralls.

La interacció amb la visualització és exactament igual que en el cas de visualitzar models simples. Hi ha els mateixos tipus d'interacció i les mateixes tecles. En aquest cas els 2 volums es mouen alhora. Podem veure el resultat d'haver mogut la càmera i el model a la Figura 6.32.

En el moment de definir les propietats de visualització d'un volum en un mirall determinat, l'única diferència és que cal seleccionar el volum mitjançant el quadre combinat que hem comentat anteriorment. Ara el quadre combinat té una entrada per cada volum, com podem veure a la Figura 6.31.

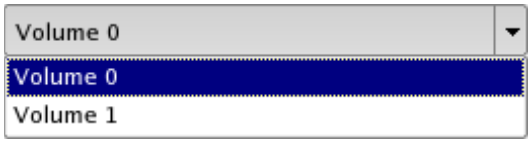


Figura 6.31: Quadre combinat de selecció de volum.

En la visualització de models fusionats sí que té sentit fer que un volum sigui invisible en alguns miralls. Per exemple, podem fer que al volum central i al mirall 3 es vegin tots dos volums, al mirall 1

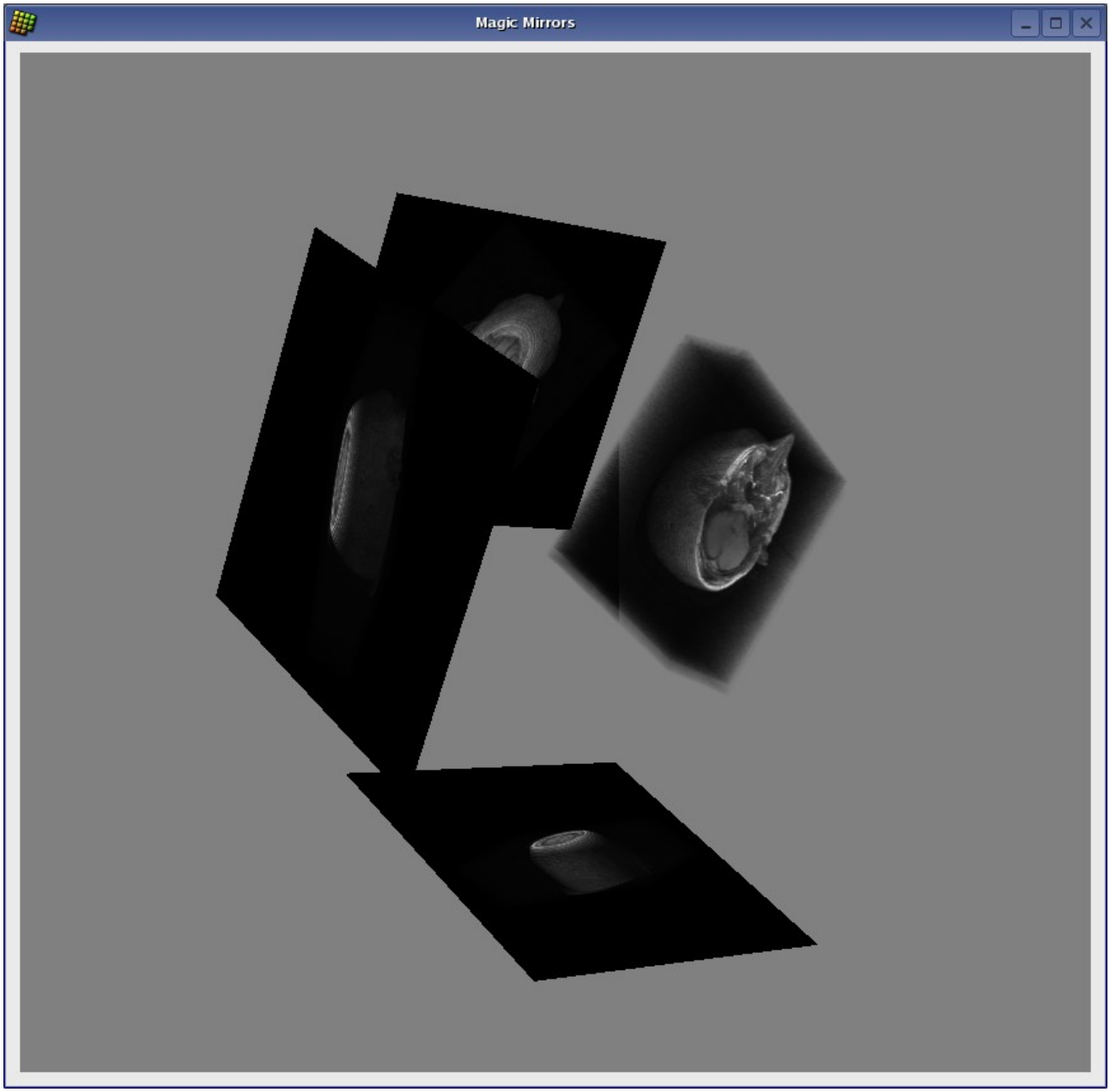


Figura 6.32: Miralls Màgics després d'haver girat la càmera i el model.

es vegi només el volum 0 i al mirall 2 es vegi només el volum 1. Per fer-ho cal desmarcar el quadre de verificació "Visible" del volum 1 al mirall 1 i el del volum 0 al mirall 2. Amb això obtenim una imatge com la de la Figura 6.33. No s'observen gaires diferències perquè en aquest cas els dos volums són molt semblants.

Una característica pròpia de la visualització de models fusionats és la prioritat de visualització. Això fa referència a l'ordre en que es dibuixen els volums, de manera que l'últim que es dibuixa es veu sempre per sobre de l'altre, i per tant podem considerar que té "prioritat". Els volums es dibuixen en l'ordre en què són afegits al *renderer*; per tant, els volums s'haurien d'afegir seguint un ordre de menys a més prioritat. Com que la interfície no permet definir la prioritat i els volums s'afegeixen i s'esborren dels *renderers* diverses vegades (quan es canvia la visibilitat d'un volum o s'interacciona

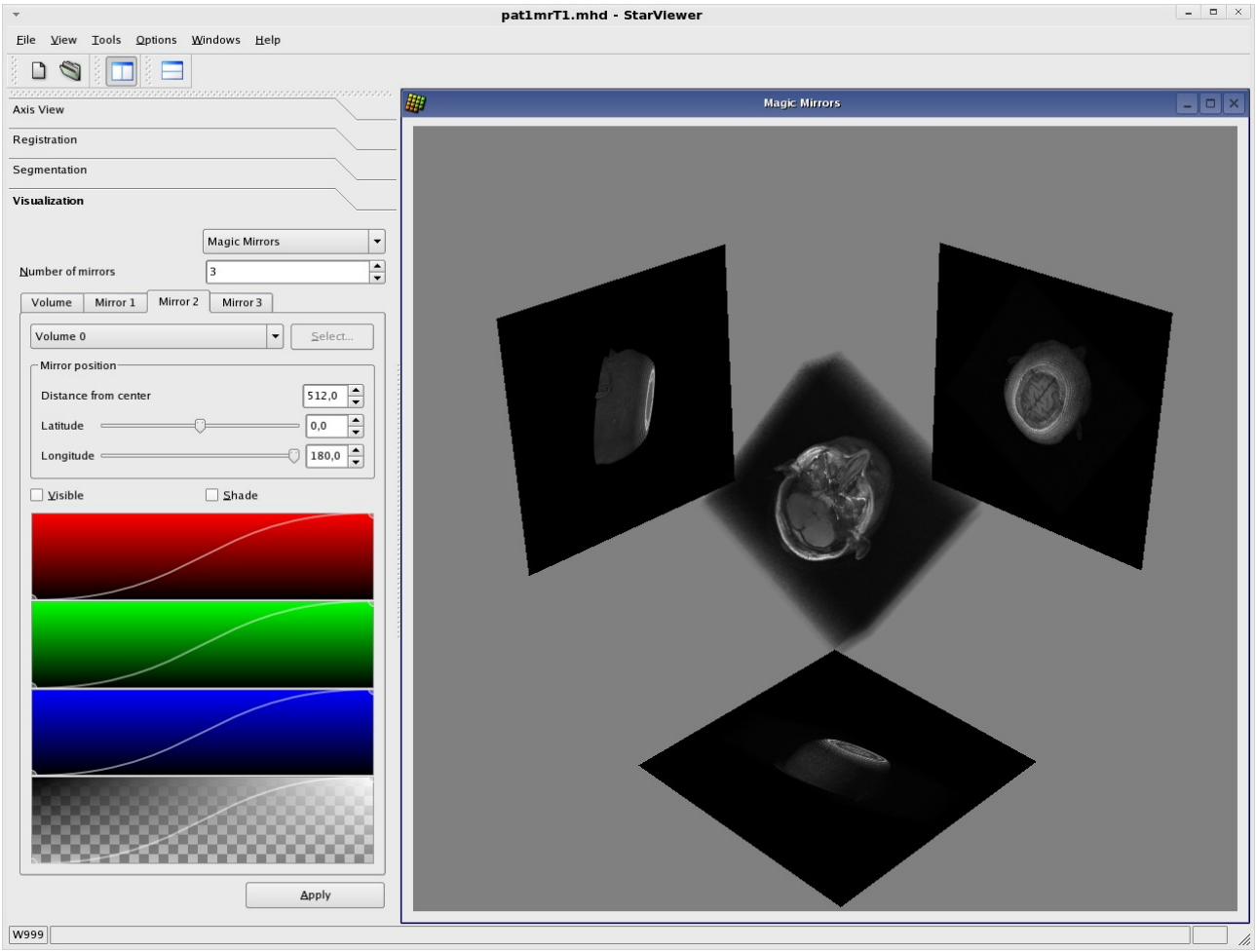


Figura 6.33: Miralls Màgics: s'ha tret el volum 0 del mirall 1 i el volum 1 del mirall 2.

amb el model), l'ordre en què es dibuixen els models va canviant mentre s'utilitza l'aplicació. No obstant, sabent això es pot arribar a definir la prioritat jugant amb les visibilitats, fent visible en últim lloc el volum que volem que sigui prioritari. Però la solució ideal seria implementar la definició de prioritats.

I aquí acaben les diferències entre la visualització de models simples i de models fusionats. A les imatges següents hi ha diferents exemples de configuracions de la visualització del model fusionat.

A la Figura 6.34 al centre hem fet invisible el volum 1 hem canviat la funció de transferència del volum 0.

A la Figura 6.35 hem assignat una nova funció de transferència al volum 0 del mirall 1.

A la Figura 6.36 hem assignat una nova funció de transferència al volum 1 del mirall 2.

A la Figura 6.37 al mirall 3 hem fet invisible el volum 0 i hem canviat la funció de transferència del volum 1. A més a més hem apropat tots els miralls al volum, aconseguint d'aquesta manera una millor visualització.

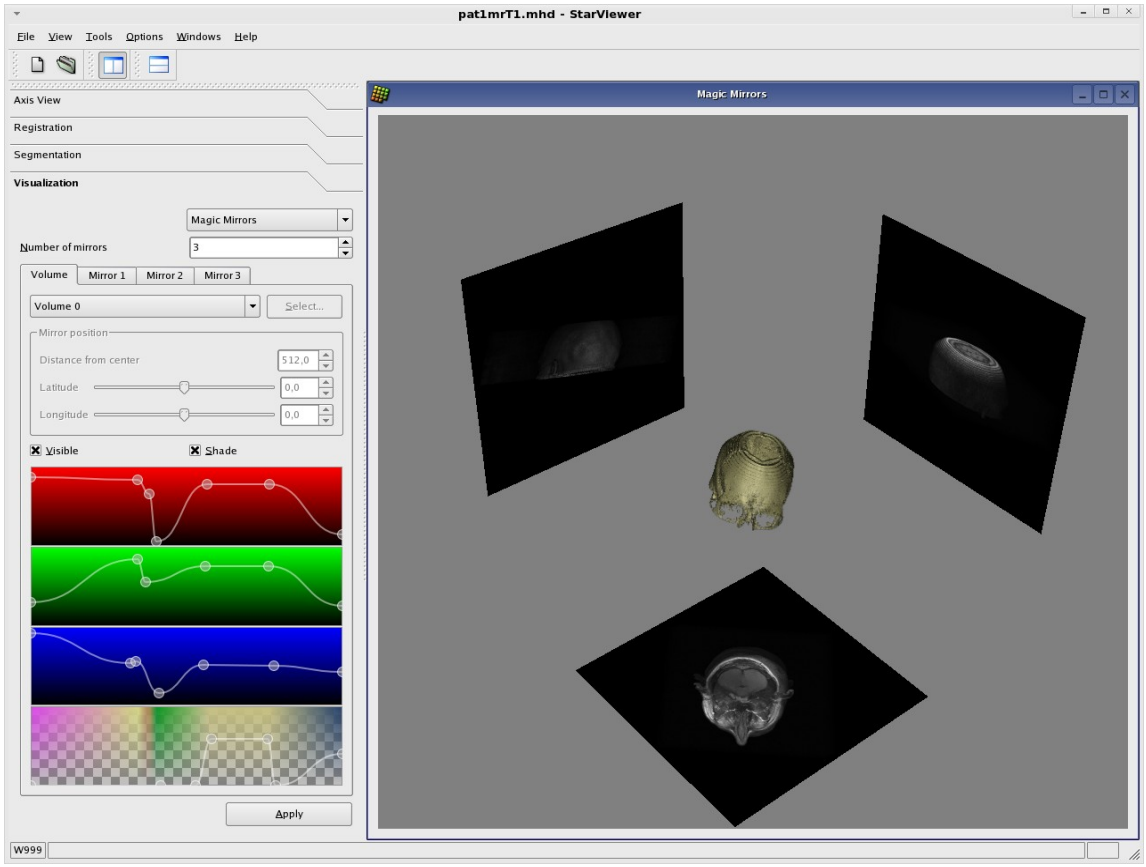


Figura 6.34: Miralls Màgics amb el model fusionat "pat1ct.mhd" + "pat1mrT1.mhd".

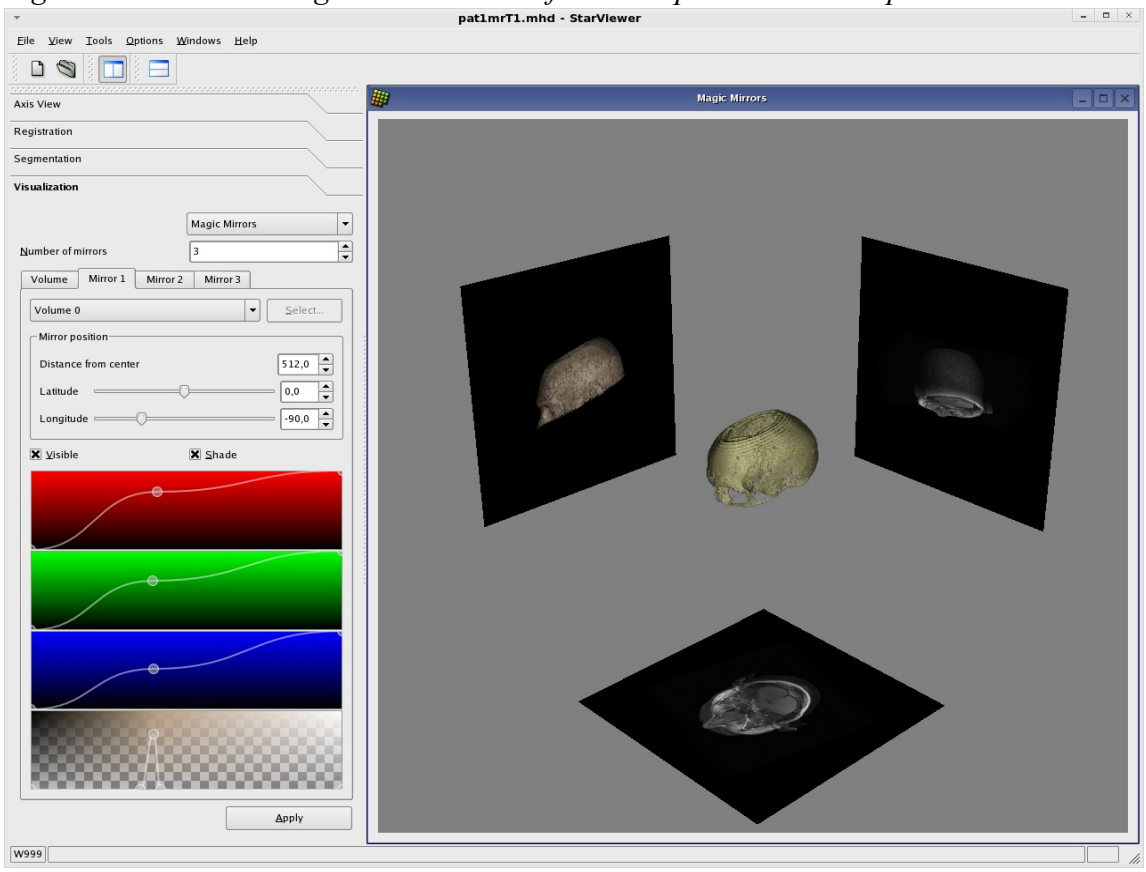


Figura 6.35: Miralls Màgics amb el model fusionat "pat1ct.mhd" + "pat1mrT1.mhd".

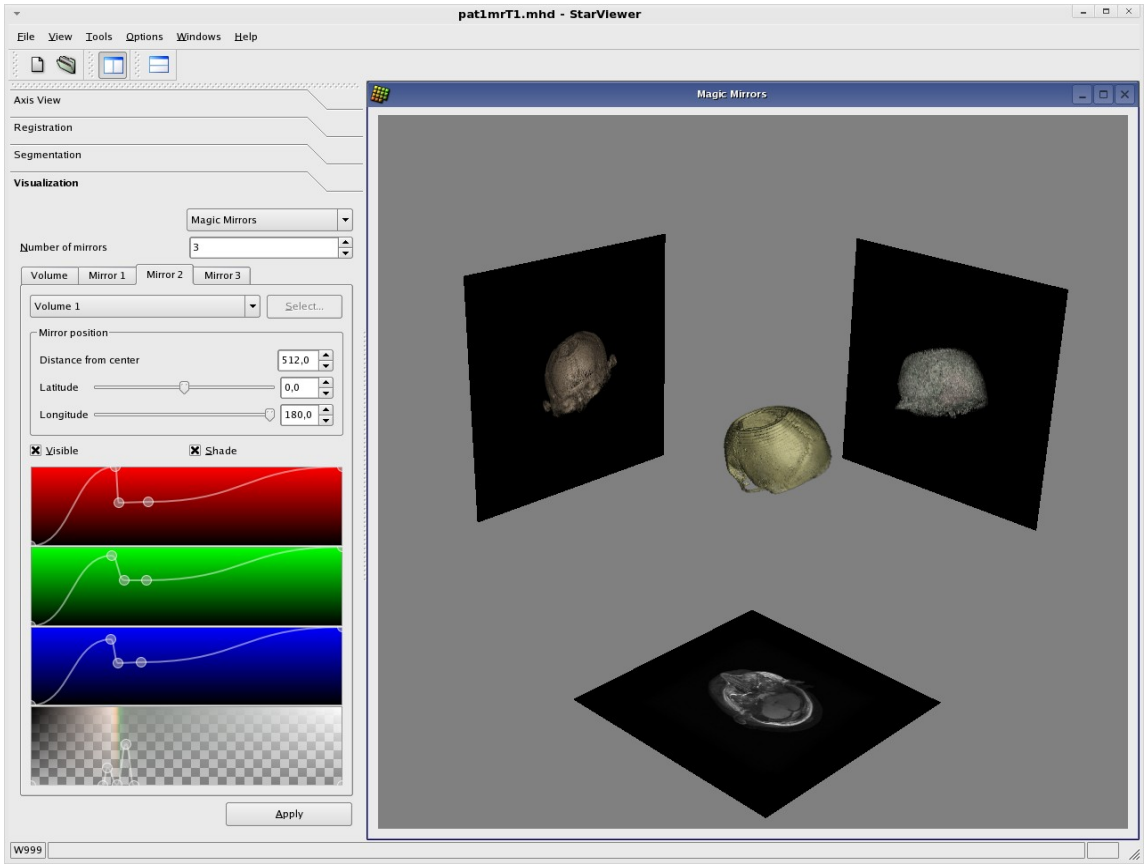


Figura 6.36: Miralls Màgics amb el model fusionat "pat1ct.mhd" + "pat1mrT1.mhd".

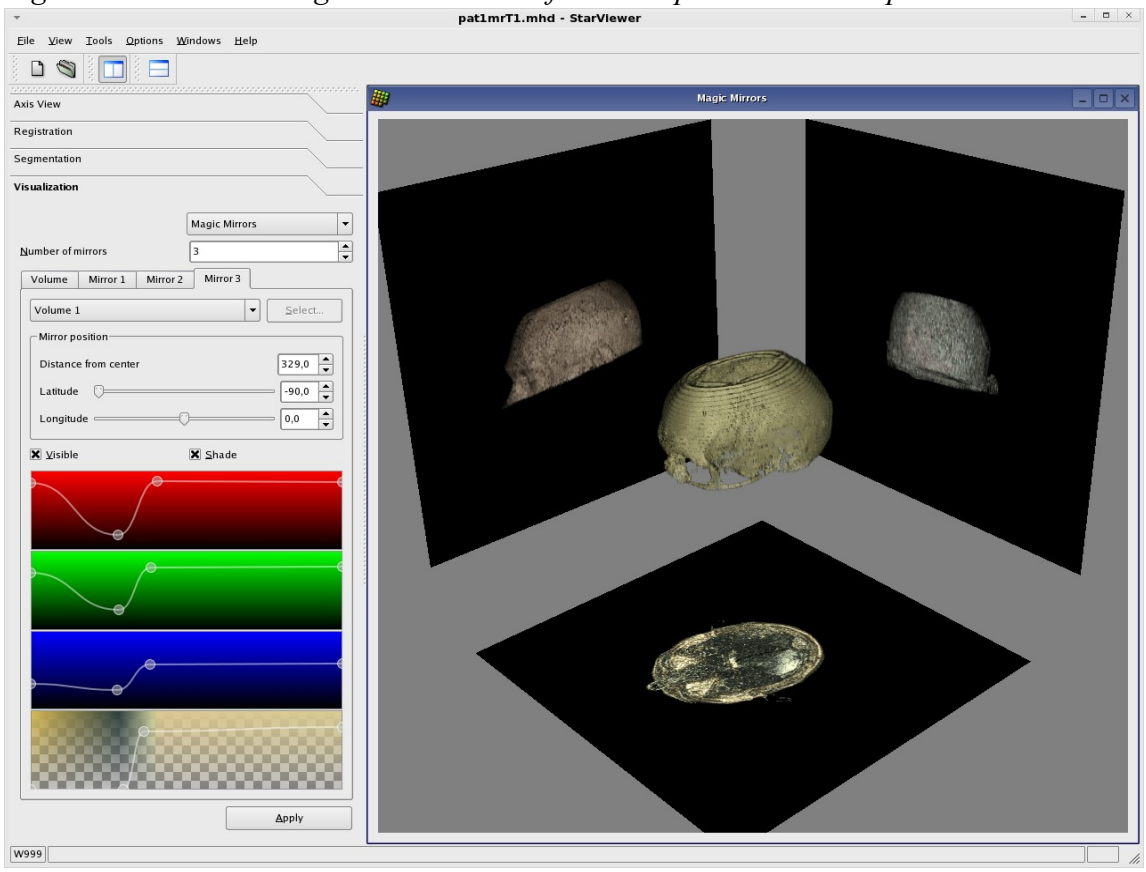


Figura 6.37: Miralls Màgics amb el model fusionat "pat1ct.mhd" + "pat1mrT1.mhd".

6.3 Optimal Viewpoint

Veiem ara el funcionament del mètode de selecció del punt de vista òptim, que hem anomenat *Optimal Viewpoint*.

Un cop hem obert un model (en aquest exemple "pat1ct.mhd") i hem accedit a la part del Punt de Vista Òptim, el pas següent més lògic és seleccionar-lo per a la visualització. Això ho fem prement el botó "Select..." i triant el volum al quadre de diàleg que apareix, que té la mateixa aparença que el dels Miralls Màgics. La diferència és que si hi hagués més d'un volum obert només permetria seleccionar-ne un.

Un cop seleccionat el volum és el moment de definir els paràmetres de la segmentació. L'aparença de la interfície sencera és la de la Figura 6.38. En aquest cas es poden definir tots els paràmetres fins i tot abans de seleccionar el volum, però és més lògic fer-ho en aquest ordre.

Ara comentarem els elements de la interfície per introduir els paràmetres de la segmentació.

Són un total de 6 paràmetres, que es poden veure a la Figura 6.39:

- "Iterations" és el nombre d'iteracions de l'algorisme genètic. El mínim és 1 i el màxim 500. Com més iteracions es facin més probabilitats hi ha de trobar una millor segmentació durant l'algorisme genètic, però també es farà més llarg el procés de segmentació.

- "Block length" és la longitud de bloc que es farà servir per calcular l'*excess entropy*. El mínim és 2 i el màxim 6. Com més gran més precís serà el resultat del càlcul, però també caldrà construir uns histogrames més grans.

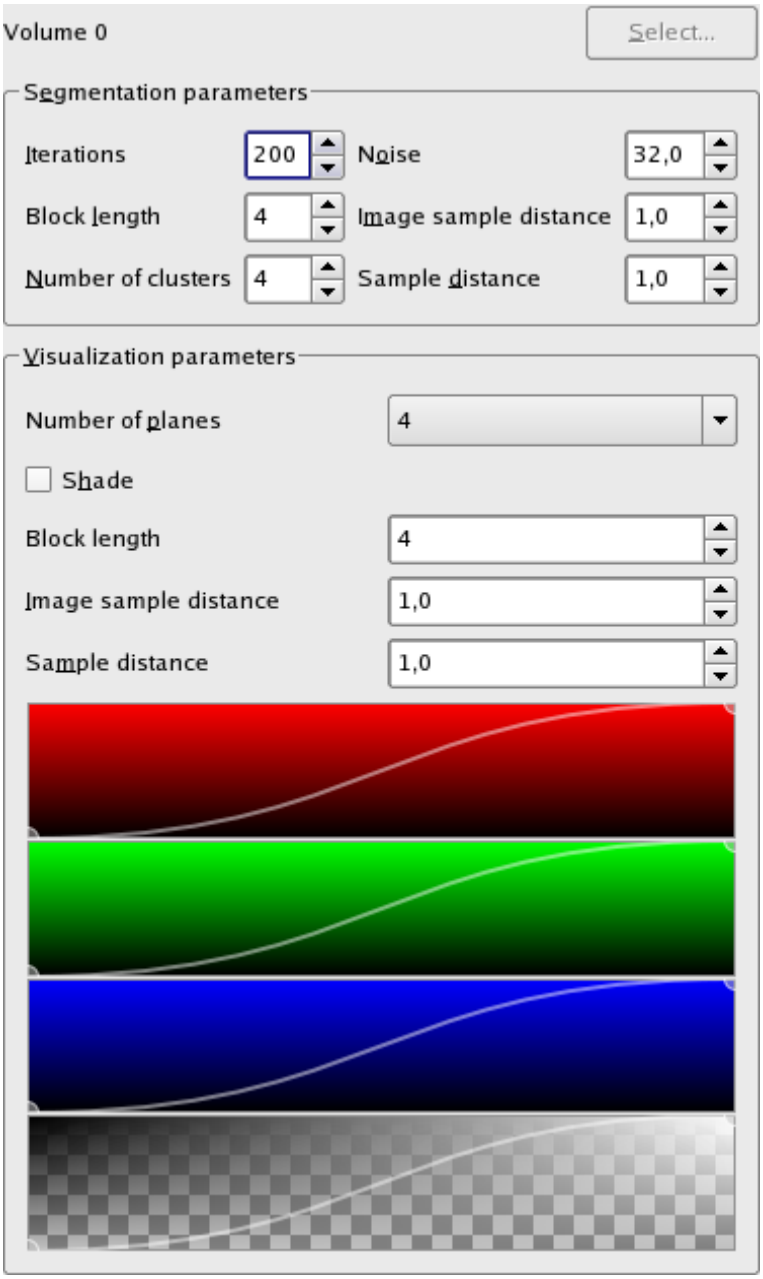


Figura 6.38: Opcions de configuració de la selecció del punt de vista.

- "Number of clusters" és el nombre de clústers en que s'ha de segmentar el model. El mínim és 2 i el màxim 6. Ha de ser igual al nombre d'estructures diferents del model, comptant l'espai buit.

- "Noise" és la quantitat màxima que es pot sumar o restar a un dels límits de la segmentació durant l'algorisme genètic. El mínim és 0.1 i el màxim 128.

- "Image sample distance" és la distància entre els raigs que es llançaran per agafar mostres per al càlcul de l'*excess entropy*. El mínim és 0.1 i el màxim 512. Com més petita més probabilitat hi haurà de detectar estructures petites, però també serà més lent.

- "Sample distance" és la distància entre les mostres que s'agafaran a cada raig. El mínim és 0.1 i el màxim 512. Com més petita més probabilitat hi haurà de detectar estructures petites, però també serà més lent.

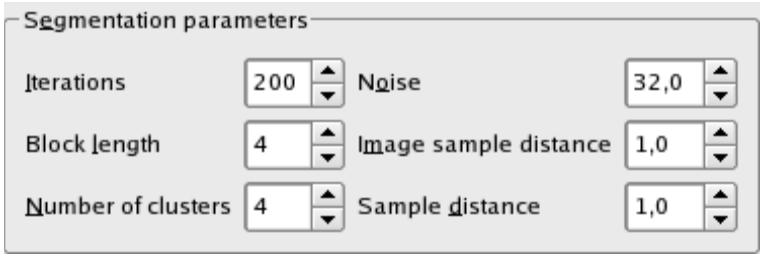


Figura 6.39: Detall dels paràmetres de la segmentació.

Un vists els paràmetres fem una segmentació de prova amb els paràmetres següents: *iterations* = 380, *block length* = 5, *number of clusters* = 6, *noise* = 13.5, *image sample distance* = 4.5, *sample distance* = 1.1. Després de prémer el botó "Apply" apareix una finestra de visualització que servirà per fer els *ray castings* necessaris durant la segmentació (Figura 6.40). Al cap d'uns minuts la finestra es tanca i apareix la visualització de la Figura 6.42. En aquest cas hem mogut la càmera perquè es vegi millor la visualització, i també hem mogut la finestra de resultats cap a un racó i la finestra principal al davant.

La visualització s'ha fet amb els paràmetres per defecte i s'ha aplicat una funció de transferència automàtica i aleatòria, però definida de manera que cada clúster quedi pintat d'un color diferent. Aquesta és la funció de transferència ajustada que es calcula automàticament després de fer la segmentació. Mirant l'editor de la transferència podem veure la forma d'aquesta funció de transferència ajustada, que té una forma esglaonada. Cada tram correspon a un clúster. El primer tram sempre té una opacitat molt baixa perquè normalment correspon a l'espai buit. En aquest cas hem tingut mala sort i l'espai buit ha quedat dividit en dues parts, i a la segona se li ha assignat una opacitat alta, però després podrem modificar aquesta funció de transferència.

També ha aparegut una finestra amb els resultats del càlcul de l'*excess entropy* de cada pla (Figura 6.41). Per saber a quina posició correspon cada pla podem alçar la seva finestra i comparar-la amb la visualització principal. En aquest cas veiem el que té *excess entropy* màxima és el pla 2, que és el que està a sobre i una mica per darrere del pla (Figura 6.43).

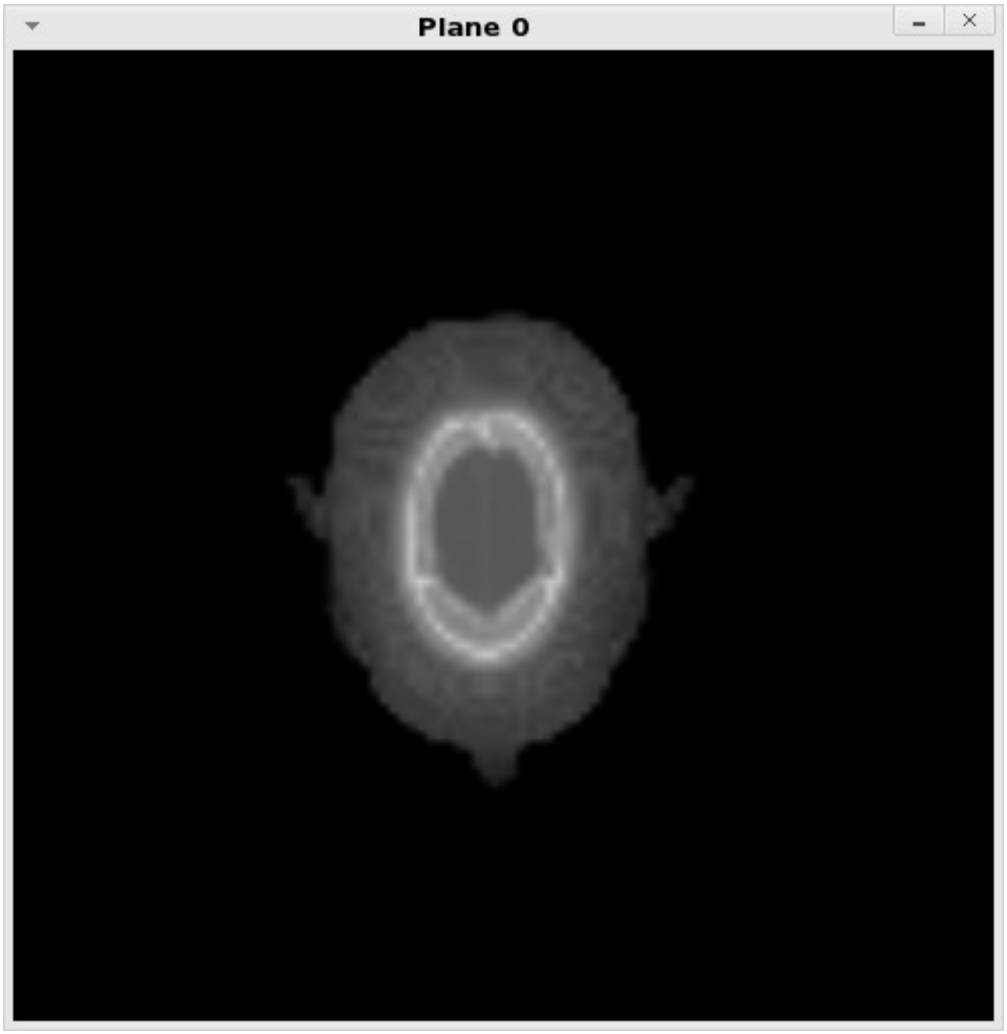


Figura 6.40: Finestra de visualització per fer la segmentació.

Per acabar amb el tema de la segmentació hem de destacar que els controls de la segmentació han quedat inhabilitats perquè aquesta ja s'ha fet i que el model segmentat es mostra als plans, mentre que el volum central continua sent sense segmentar.

Figura 6.41: Finestra de resultats.

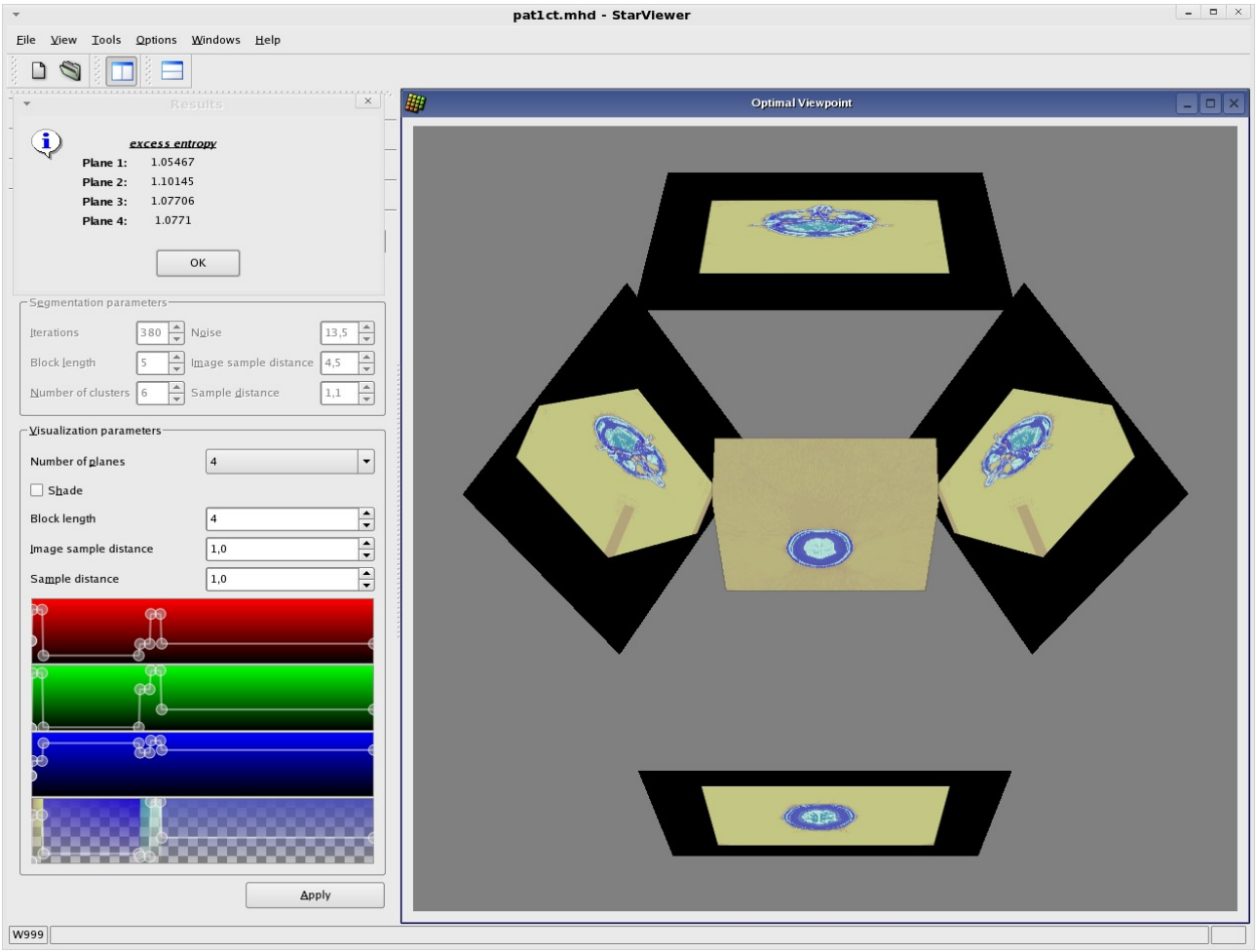


Figura 6.42: Visualització i resultats després de la segmentació.

Podem interaccionar amb la finestra de visualització principal de la mateixa manera que en els Miralls Màgics, amb els mateixos controls del ratolí i del teclat. En aquest cas si girem el model forçarem que es torni a calcular l'*excess entropy* de cada pla quan es faci la seva visualització. Això serà quan es posi la seva finestra al davant o en prémer el botó "Apply".

Ara comentarem els elements de la interfície per introduir els paràmetres de la visualització i el càlcul de l'*excess entropy* de cada pla. Aquests paràmetres són els de la Figura 6.45.

Primer de tot hi ha un quadre combinat (Figura 6.44) que permet triar el nombre de plans entre 4, 6, 8, 12 i 20. Com més plans posem més probabilitats hi haurà de trobar un punt de vista millor, però també cal tenir en compte el cost de fer cada visualització i calcular la seva *excess entropy*.

A sota seu podem trobar un quadre de verificació anomenat "Shade", que té el mateix efecte que als Miralls Màgics: aplica un ombrejat a la visualització. Aquest ombrejat no afecta l'*excess entropy*.

Després tenim tres paràmetres que ja hem vist a la part de segmentació. Són "Block length", "Image sample distance" i "Sample distance". El primer afecta només al càlcul de l'*excess entropy* amb els mateixos efectes que durant la segmentació, i els altres dos afecten a la qualitat de la visualització i a

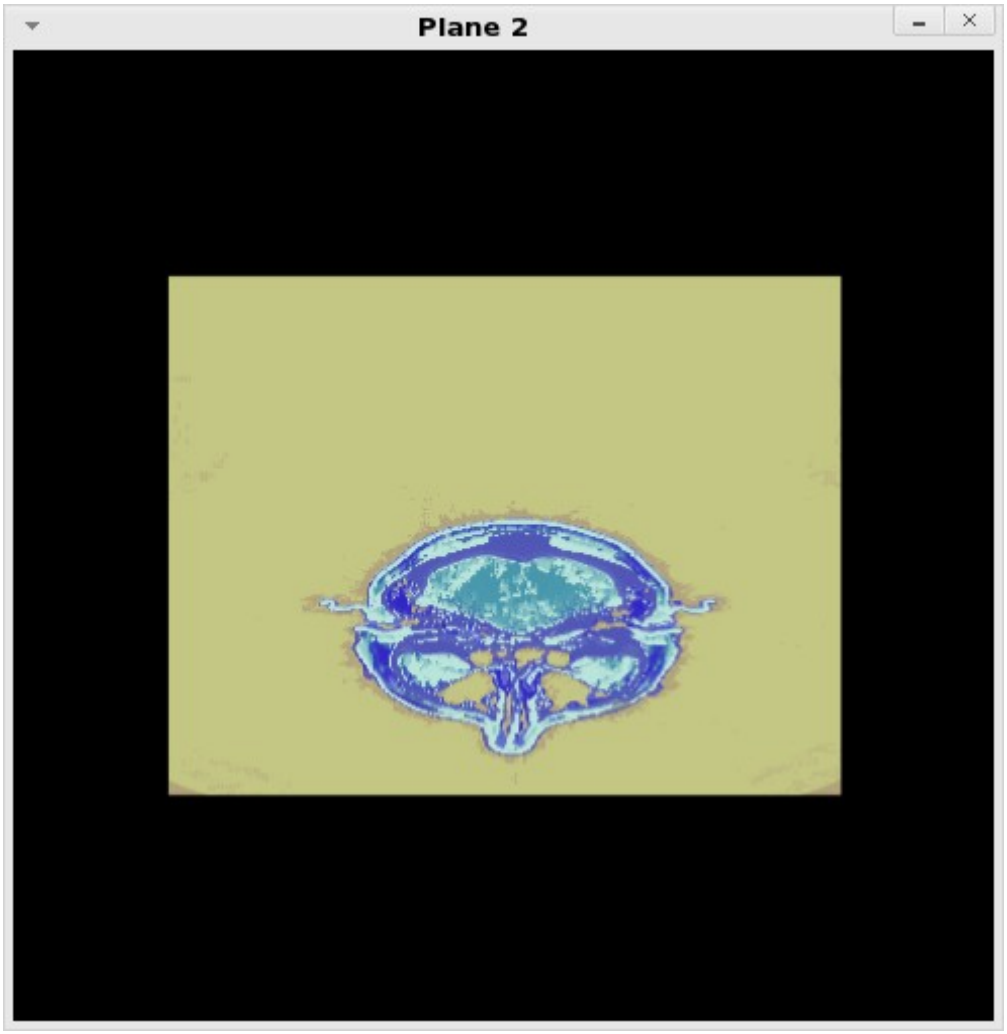


*Figura 6.43: Visualització del pla amb l'*excess entropy *més gran.*

l'*excess entropy*, també amb els mateixos efectes que durant la segmentació. Un canvi a qualsevol dels tres forçarà que es torni a calcular l'*excess entropy* de cada pla.

Finalment hi ha l'editor de la funció de transferència, que funciona exactament igual que en el cas dels Miralls Màgics. La funció de transferència no afecta l'*excess entropy*. No serveix de res configurar la funció de transferència abans de segmentar el model perquè sempre s'assigna la funció de transferència ajustada quan s'acaba la segmentació. A partir d'aquesta funció és més fàcil definir un color diferent per cada clúster: només cal moure els punts verticalment. Per exemple, podem fer

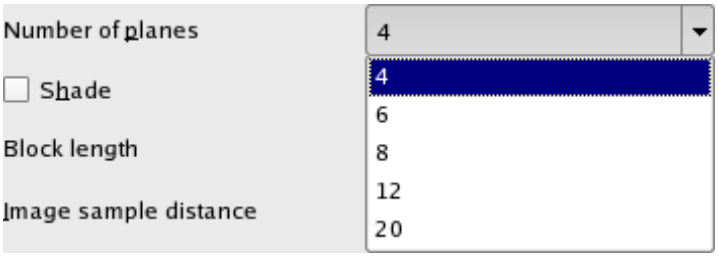


Figura 6.44: Quadre combinat per triar el nombre de plans.

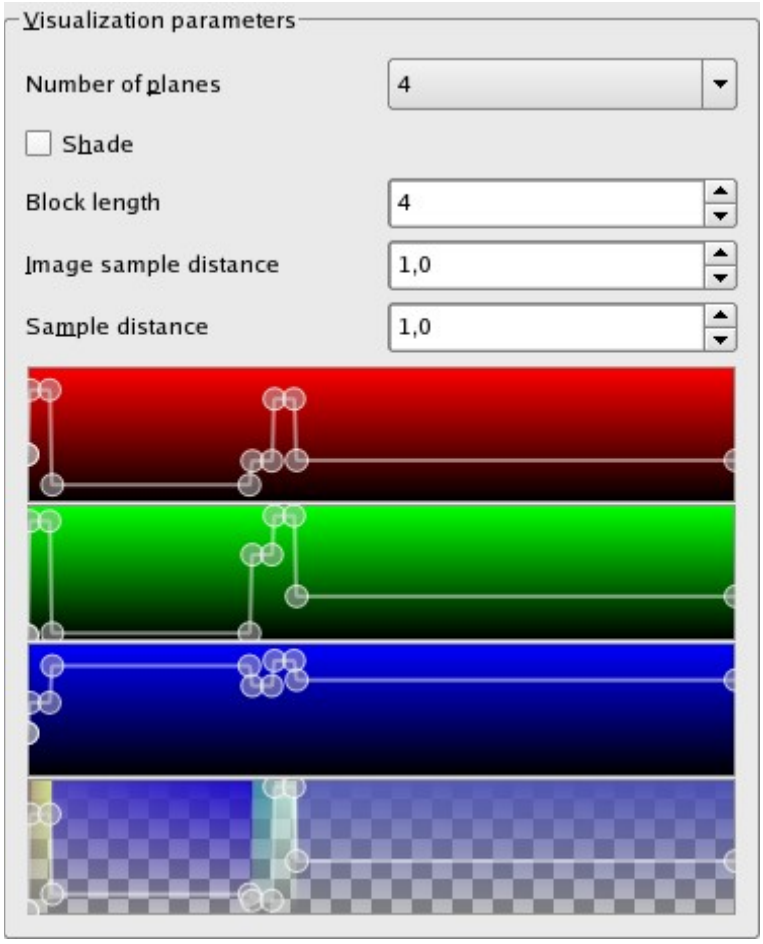


Figura 6.45: Detall dels paràmetres de la visualització.

transparent el segon clúster, que correspon a espai buit i és el que està pintat de color marró clar, obtenint la visualització de la Figura 6.46.

Les imatges següents mostren exemples de segmentacions i resultats amb diferents models i paràmetres.

A la Figura 6.47 hem girat el model i ara l'*excess entropy* màxima és al pla 4.

A la Figura 6.48 hem canviat les opcions de visualització i hem girat el model i ara l'*excess entropy* màxima torna a ser al pla 2.

A la Figura 6.49 apliquem el mètode al model artificial "m0.mhd", on hi ha 4 cubs amb diferents valors de propietat; l'*excess entropy* màxima és al pla 1, al davant del model.

A la Figura 6.50 apliquem el mètode al model artificial "m1.mhd", on hi ha 4 paral·lelepípedes amb diferents valors de propietat; aquest cop ho fem amb 6 plans i l'*excess entropy* màxima és al pla 1, al davant del model. Amb els dos models artificials la segmentació ha estat correcta.

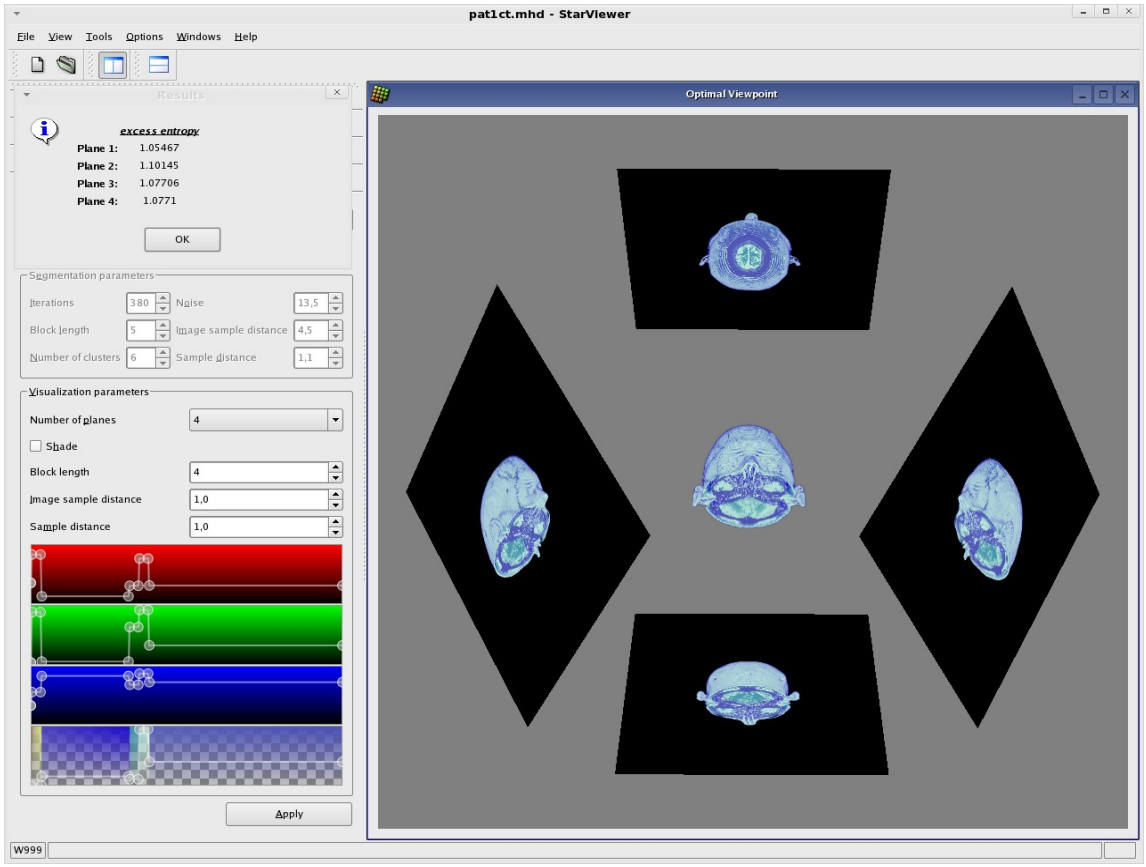


Figura 6.46: S'han fet totalment transparents els dos primers clústers.

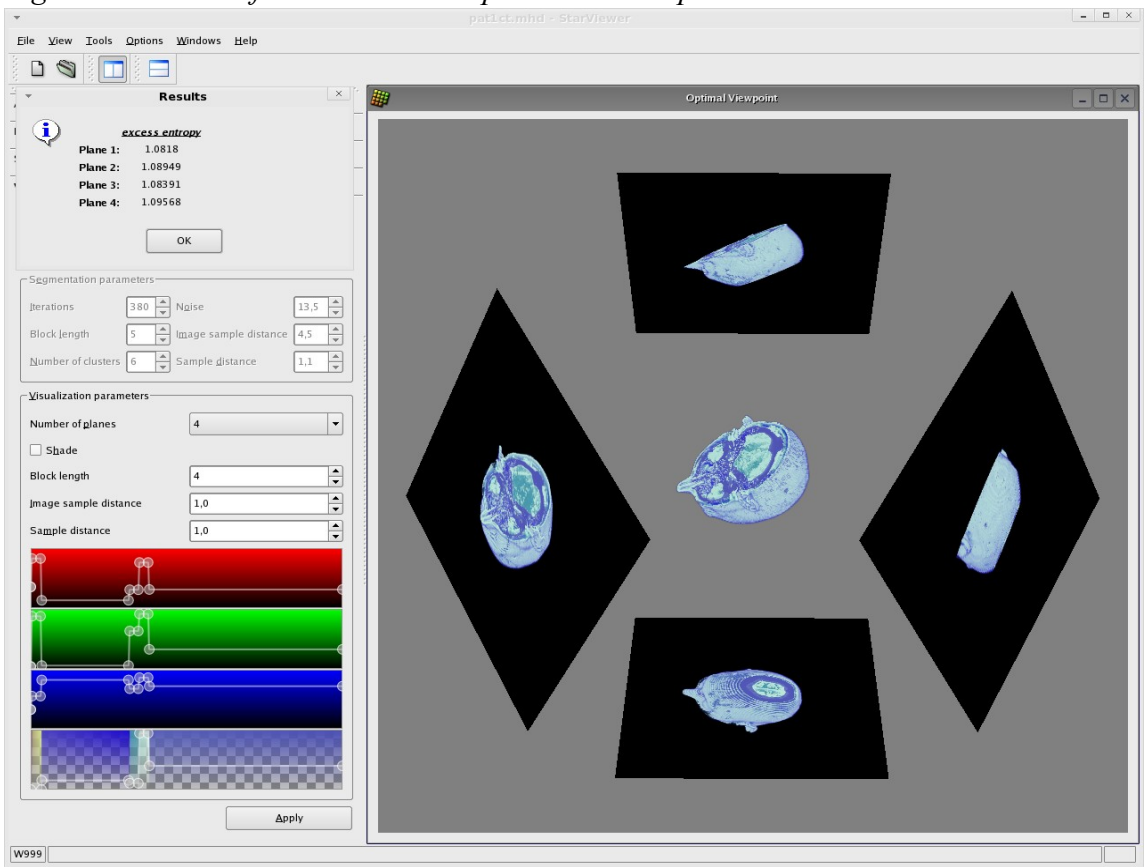


Figura 6.47: "pat1ct.mhd": s'ha girat el model.

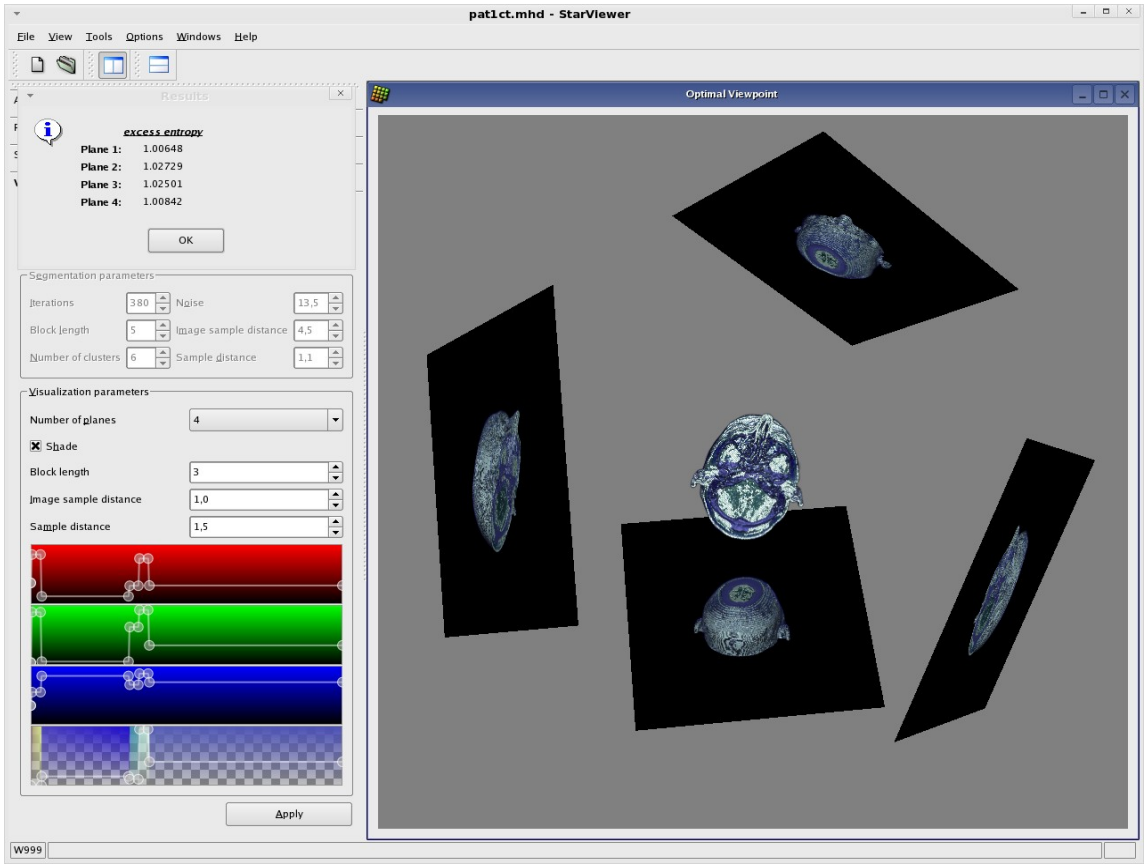


Figura 6.48: "pat1ct.mhd": s'han canviat els paràmetres de visualització i s'ha girat el model.

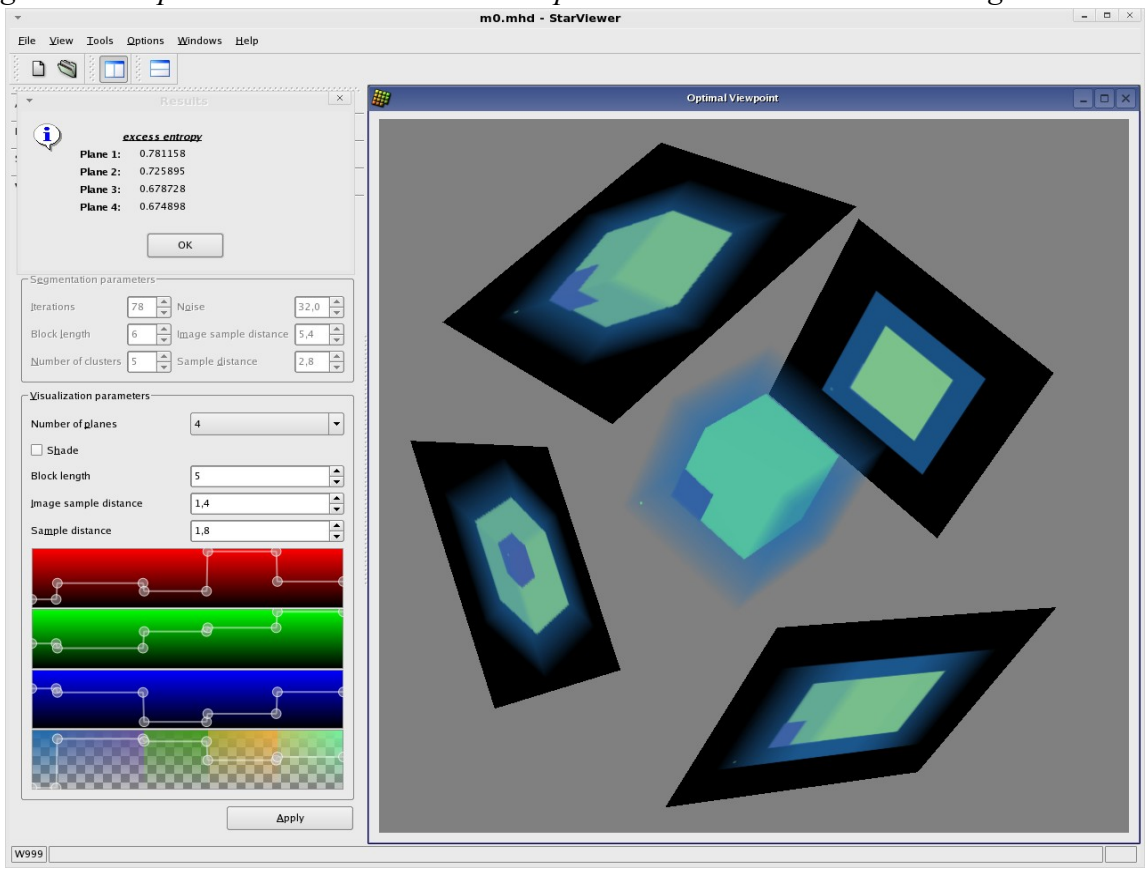


Figura 6.49: Selecció del punt de vista òptim amb "m0.mhd".

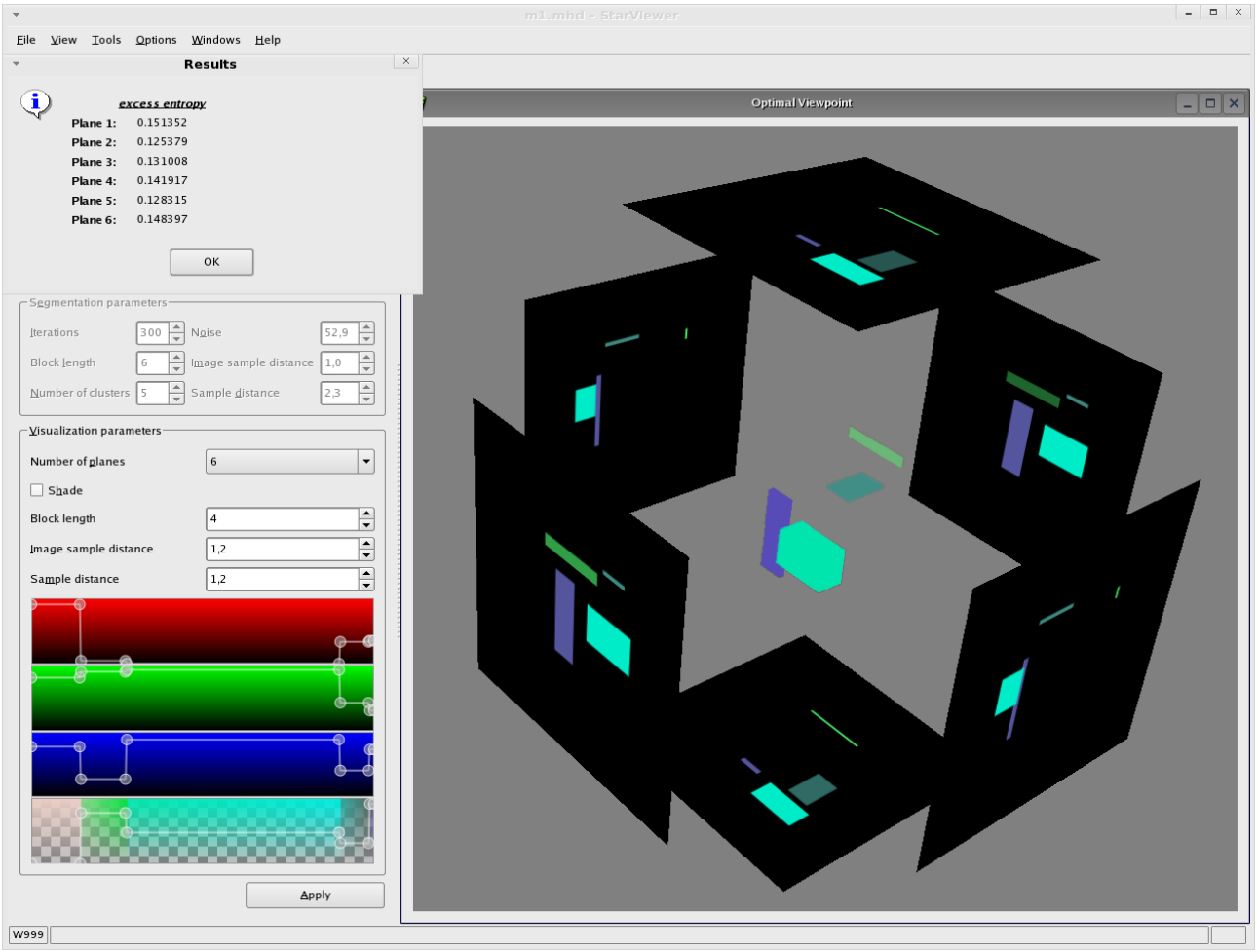


Figura 6.50: Selecció del punt de vista òptim amb "m1.mhd", amb 6 plans.

7 Avaluació dels resultats

7.1 Entorn de proves

Les proves les hem fet amb un ordinador amb les característiques següents:

- Processador Intel Pentium 4 2.8 GHz
- 1024 MB DDR RAM (2 × 512 MB)
- Targeta gràfica ATI Radeon 9000 Series 128 MB RAM
- Sistema operatiu GNU/Linux Fedora Core 5
- Kernel 2.6.17
- Sistema de finestres X.Org 7.0
- Entorn d'escriptori GNOME 2.14 (principal)
- Entorn d'escriptori KDE 3.5.4 (instal·lat)
- Resolució de pantalla de 1280 × 1024
- Compilador GCC 4.1.1
- Qt 4.1.4
- ITK 2.6.0
- VTK 5.0.0

També cal dir que els temps d'execució que sortiran a l'apartat següent són per la versió de publicació de l'aplicació (no la de depuració), i això fa que siguin més petits.

7.2 Proves i resultats

En aquest apartat mostrarem alguns resultats obtinguts amb les dues tècniques implementades amb diferents models i paràmetres. També mostrarem alguns temps de referència aproximats, com el temps d'actualitzar un mirall, actualitzar un pla, segmentar un model, etc. A l'apartat següent farem la valoració sobre el rendiment de l'aplicació i els resultats obtinguts.

7.2.1 Miralls Màgics

Model simple "t1_n7_rf0.mhd"

Fem la visualització amb els paràmetres següents:

- 3 miralls.
- Mirall 1 a la posició (274.4, 0, -90).
- Mirall 2 a la posició (274.4, 0, 180).
- Mirall 3 a la posició (274.4, -90, 0).
- L'ombrejat està activat al volum central i als 3 miralls.

La visualització obtinguda és la de la Figura 7.1.

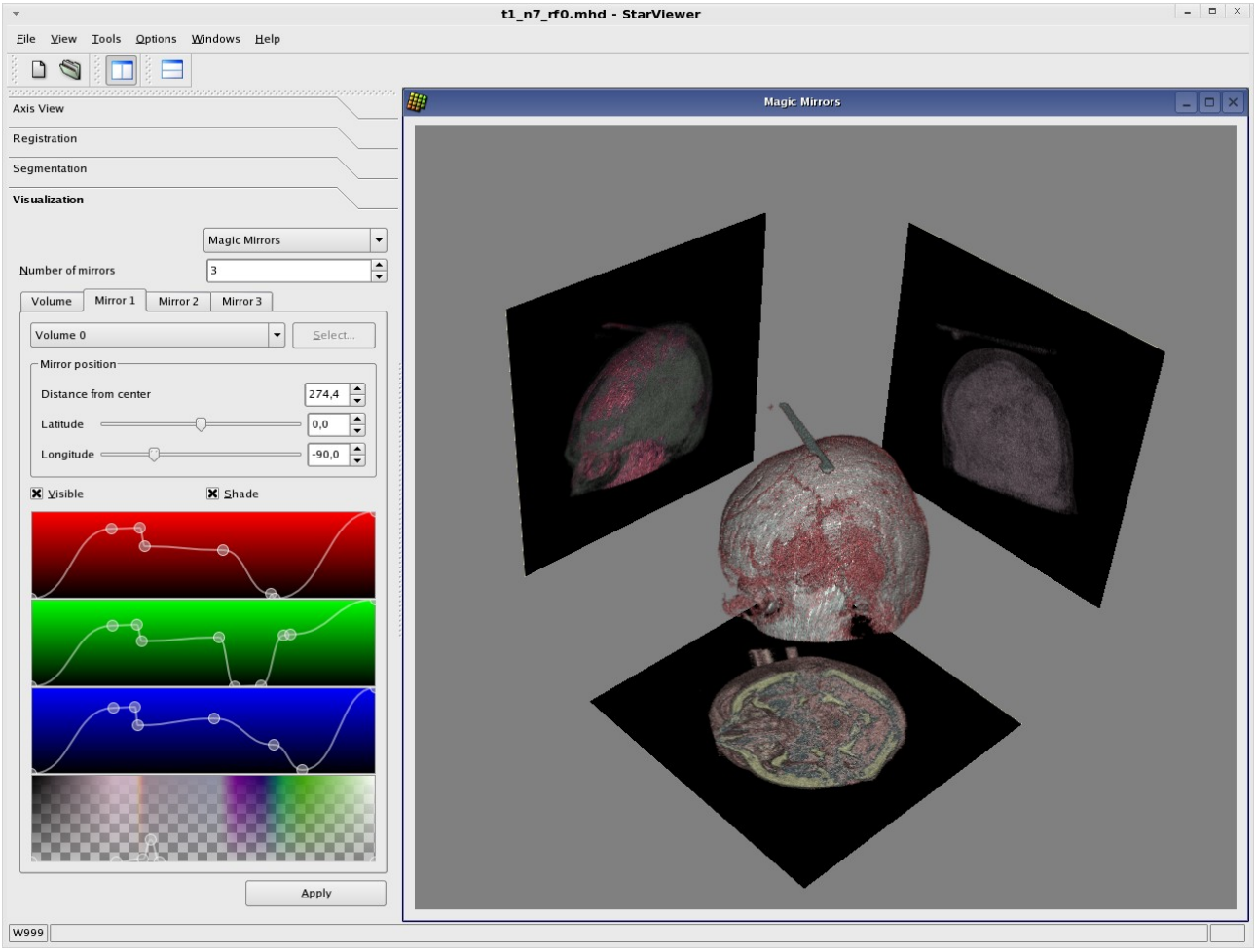


Figura 7.1: Miralls Màgics amb el model simple "t1_n7_rf0.mhd".

Els temps de referència per aquest model són els següents:

- Temps d'actualització dels 3 miralls: 3.57 segons. Mitjana: 1.19 segons.
- Temps de resposta en interaccionar amb el model: aproximadament 1 segon.

Model fusionat "pat1ct.mhd" + "pat1mrT1.mhd"

Fem la visualització amb els paràmetres següents:

- 4 miralls.
- Mirall 1 a la posició (512, 30, -150).
- Mirall 2 a la posició (512, 30, 150).
- Mirall 3 a la posició (512, -30, -150).
- Mirall 4 a la posició (512, -30, 150).
- Als miralls 1 i 3 es visualitza el primer volum. Al centre i als miralls 2 i 4 es visualitza el segon volum.
- L'ombrejat està activat al volum central i als miralls 1 i 2.

La visualització obtinguda és la de la Figura 7.2. Hem deixat els miralls 3 i 4 (els de baix) amb la funció de transferència per defecte perquè es vegi la diferència entre els dos models, que és petita.

Per al model fusionat hem mesurat més temps de referència, amb diferents combinacions d'opcions de visualització que podien afectar el temps. Els temps de referència per aquest model són els següents:

- Temps d'actualització dels 4 miralls:
  - Amb 1 volum a cada mirall i sense ombrejat: 4.64 segons. Mitjana: 1.16 segons.
  - Amb 1 volum a cada mirall i amb ombrejat: 8.76 segons. Mitjana: 2.19 segons.
  - Amb 2 volums a cada mirall i sense ombrejat: 8.88 segons. Mitjana: 2.22 segons.
  - Amb 2 volums a cada mirall i amb ombrejat: 18.32 segons. Mitjana: 4.58 segons.
- Temps de resposta en interaccionar amb el model: aproximadament 1 segon.

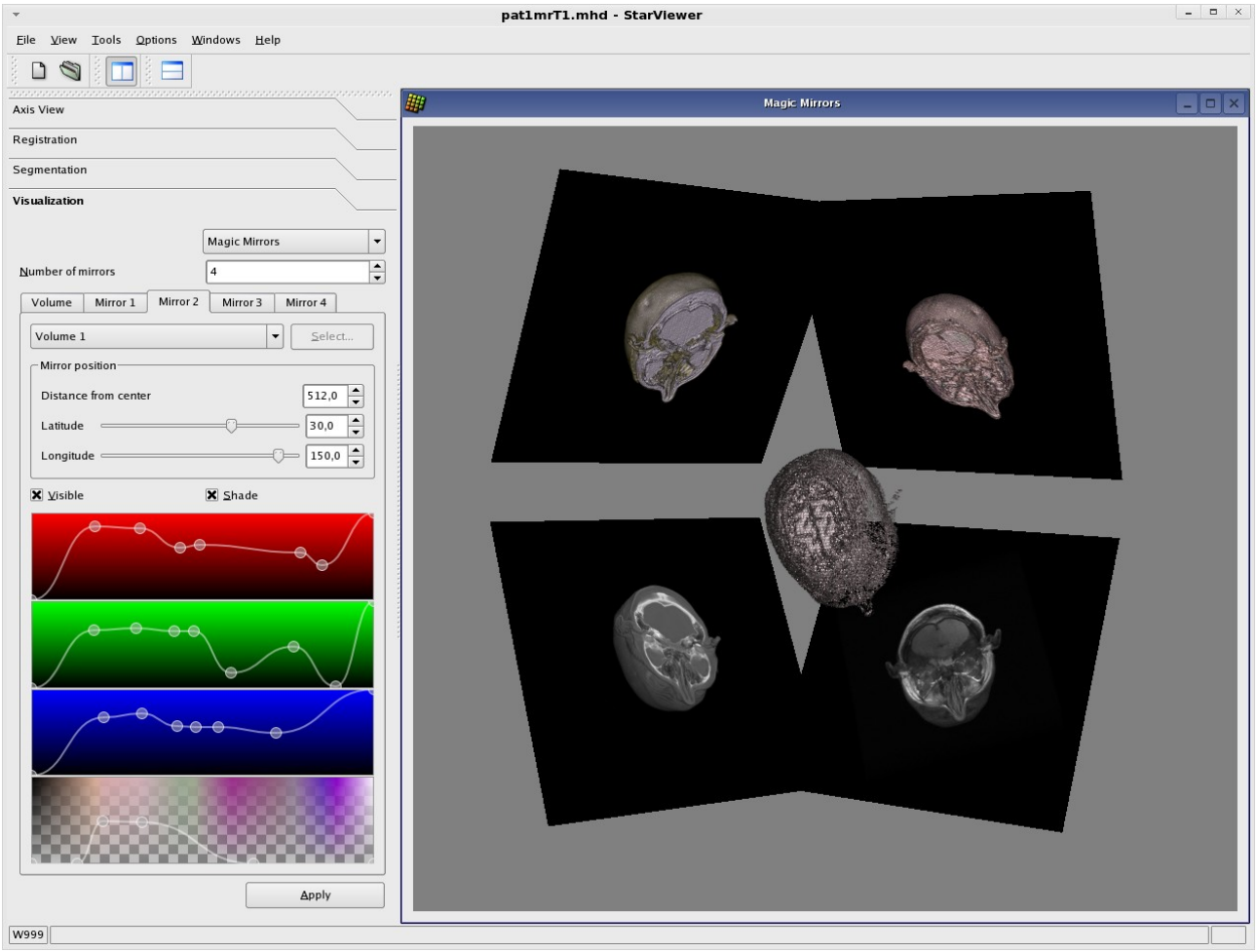


Figura 7.2: Miralls Màgics amb el model fusionat "pat1ct.mhd" + "pat1mrT1.mhd".

7.2.2 Selecció del punt de vista òptim

Model sintètic "m0.mhd"

Paràmetres de la segmentació:

- Iteracions: 400.
- Longitud de bloc: 6.
- Nombre de clústers: 5.
- Soroll: 50.
- Distància entre raigs: 1.8.
- Distància entre mostres: 2.4.

Els valors d'aquest model sintètic després d'escalar-los al rang [0,255] són 0, 25, 116, 242, 255. Els límits trobats per l'algorisme de segmentació són 5, 79, 168, 244. Els límits representen el valor màxim de cada clúster, començant pel primer (per l'últim sempre és 255). Per tant, podem dir que la segmentació ha estat correcta. Aquests límits s'han trobat a la iteració 377 de l'algorisme genètic, amb una *excess entropy* de 0.730355.

Paràmetres de la visualització:

- 4 plans.
- Ombrejat desactivat.
- Longitud de bloc: 4.
- Distància entre raigs: 1.
- Distància entre mostres: 1.

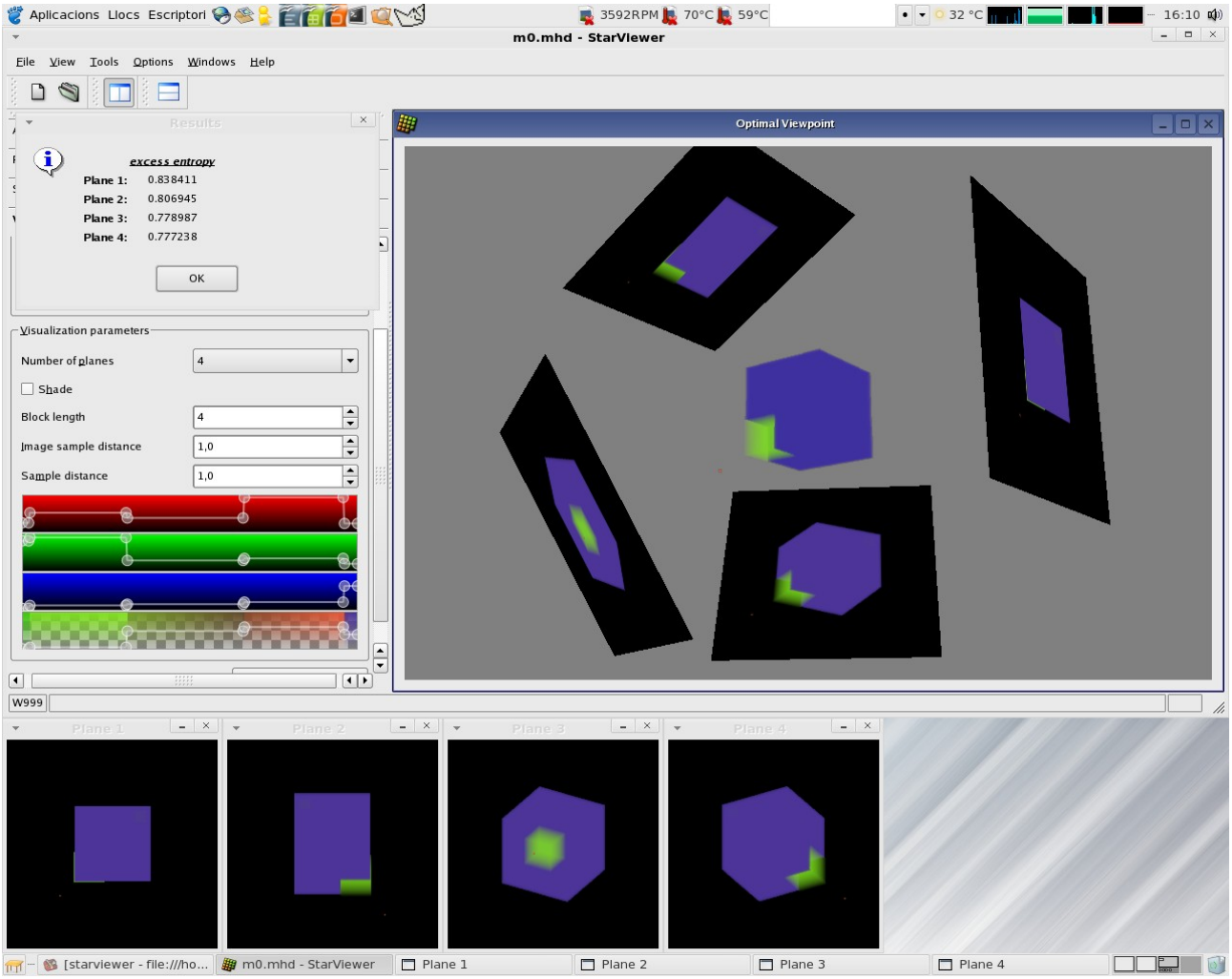


Figura 7.3: Selecció del punt de vista òptim amb el model sintètic "m0.mhd".

La visualització obtinguda és la de la Figura 7.3. En aquest cas, com que les finestres dels plans són petites les hem posat ordenades a sota de la finestra principal perquè es vegin totes alhora. La funció de transferència és la que s'ha generat automàticament. Podem veure que el pla teòricament millor (el que té una *excess entropy* més gran) és el pla 1, on el model es veu des de davant, amb una *excess entropy* de 0.838411. Només hem girat la càmera, el model continua en la seva posició inicial.

Els temps de referència per aquest model són els següents:

- Temps de segmentació: 9 minuts i 10 segons.
- Temps per actualitzar un pla sense calcular l'*excess entropy*: 6.42 segons.
- Temps per actualitzar un pla calculant l'*excess entropy*: 7.56 segons.
- Temps de resposta en interaccionar amb el model: menys d'un segon.

Model "t1_n7_rf0.mhd"

Paràmetres de la segmentació:

- Iteracions: 272.
- Longitud de bloc: 4.
- Nombre de clústers: 5.
- Soroll: 11.2.
- Distància entre raigs: 4.
- Distància entre mostres: 1.1.

Els límits trobats per l'algorisme de segmentació són 26, 69, 109, 150. Aquests límits s'han trobat durant l'algorisme d'afinament, amb una *excess entropy* d'1.32875.

Paràmetres de la visualització:

- 8 plans.
- Ombrejat desactivat.
- Longitud de bloc: 4.
- Distància entre raigs: 1.
- Distància entre mostres: 1.

La visualització obtinguda és la de la Figura 7.4. La funció de transferència és la que s'ha generat automàticament, que permet diferenciar fàcilment els diferents clústers. No obstant, es podria modificar la funció de transferència per fer-la més realista sense cap problema. Aquest cop hem girat la càmera i el model. Podem veure que el pla teòricament millor (el que té una *excess entropy* més gran) amb aquest gir és el pla 1, amb una *excess entropy* d'1.31331. En aquest cas les finestres dels plans són grans i només hem posat la del pla teòricament millor, i es veu a baix a la dreta.

Els temps de referència per aquest model són els següents:

- Temps de segmentació: 17 minuts i 8 segons.
- Temps per actualitzar un pla sense calcular l'*excess entropy*: 32.29 segons.
- Temps per actualitzar un pla calculant l'*excess entropy*: 37.74 segons.
- Temps de resposta en interaccionar amb el model: menys d'un segon.

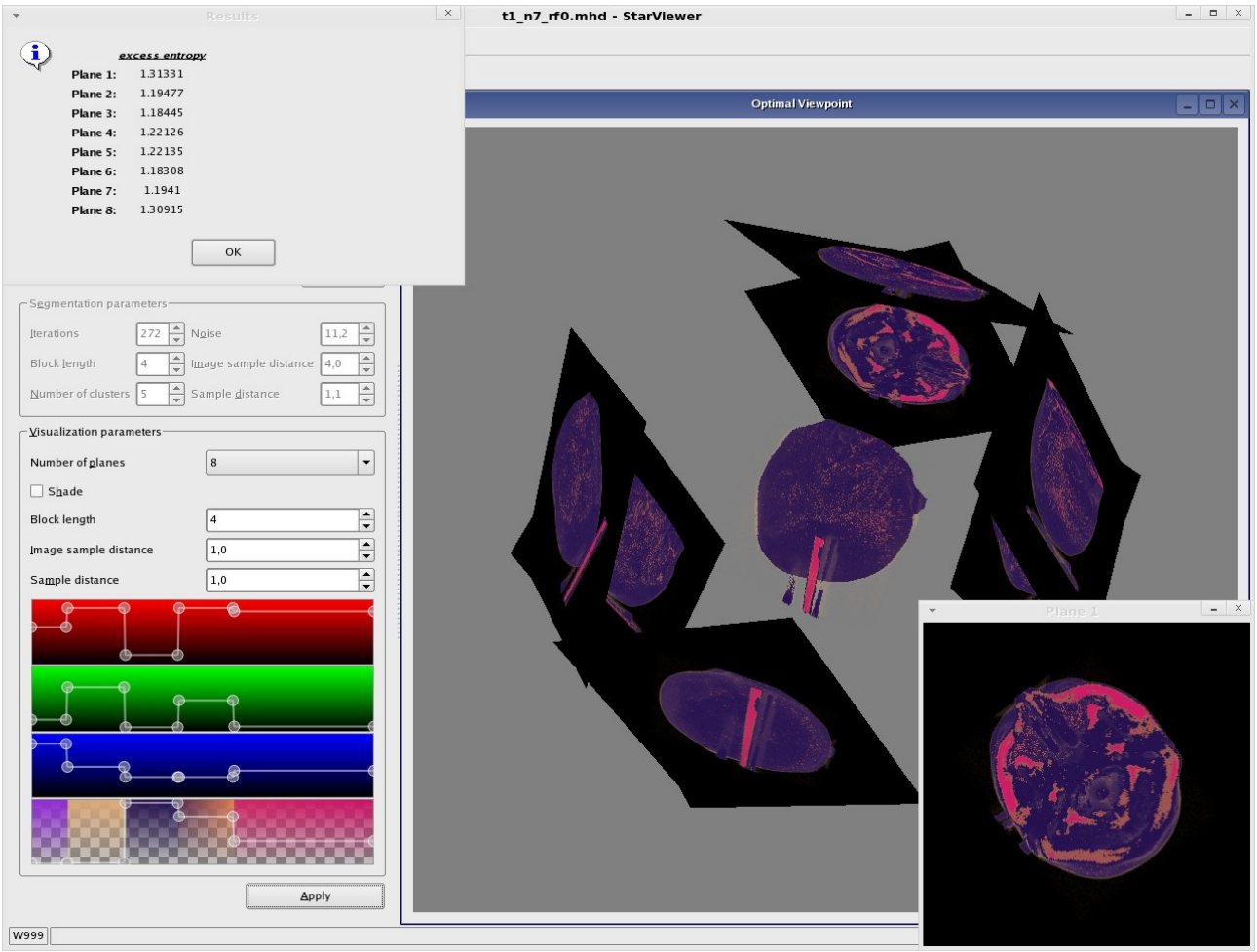


Figura 7.4: Selecció del punt de vista òptim amb el model sintètic "t1_n7_rf0.mhd".

7.3 Avaluació

Pel que fa als Miralls Màgics, podem veure que el temps d'actualitzar un mirall amb el primer model amb ombrejat és aproximadament igual que el d'actualitzar un mirall sense ombrejat amb un sol volum amb el model fusionat, i en aquest segon cas aplicant l'ombrejat gairebé es dobla el temps. La causa d'aquest fet és la grandària dels models: el segon és molt més gran que el primer. Un model més gran requereix una finestra més gran, i una finestra més gran vol dir que es llancen més raigs durant el *ray casting*. Per tant veiem que el temps de visualització d'un mirall depèn de la mida del model.

D'altra banda, amb el model registrat el temps de visualitzar dos volums és aproximadament el doble que el de visualitzar-ne un. En aquest cas els dos volums tenen la mateixa mida, i per tant veiem que el temps de visualització d'un mirall és directament proporcional al nombre de volums que es visualitzen, quan aquests són de la mateixa mida.

Acabant amb les comparacions de temps dels Miralls Màgics, el temps de visualització amb ombrejat és aproximadament el doble que sense ombrejat, en aquest cas concret. Caldria fer més mesures per saber si és sempre així.

Observant els temps individualment els podem considerar raonables. El més gran és d'uns 4 segons i mig, i això és amb el model fusionat format pels dos volums més grans dels quals disposem i aplicant un ombrejat. La majoria dels models són més petits, així que podem considerar que els temps estan prou bé. Amb un ordinador com el de proves seria suficient per fer anar l'aplicació sense gaires molèsties.

Pel que fa als temps de la selecció del punt de vista òptim, el primer que veiem és que el més costós en temps és la segmentació, com era d'esperar. Amb el model sintètic, que és petit, ha trigat una mica més de 9 minuts, i amb l'altre, d'una mida mitjana, ha trigat una mica més de 17 minuts. Tot i que són temps elevats no ens haurien de preocupar perquè la segmentació es fa només una vegada a l'inici del procés. També cal dir que el temps de segmentació depèn de molts factors, com el nombre d'iteracions de l'algorisme genètic, el nombre de clústers, el temps de fer una visualització calculant l'*excess entropy*, etc.

El temps de fer una visualització d'un pla sí que canvia molt entre els dos models, i en aquest cas sí que és només per la mida del model, perquè els paràmetres de visualització són els mateixos (el nombre de plans no importa perquè estem parlant del temps d'un pla). Amb això tornem a comprovar que el temps de visualització d'un pla depèn molt de la mida del model que s'hi visualitza.

Si comparem el temps d'actualitzar un pla sense calcular l'*excess entropy* i calculant-la, amb tots dos models veiem que calculant l'*excess entropy* el temps s'incrementa entre un 16 % i un 17 %. Per tant podem considerar que aquest és el cost afegit per calcular l'*excess entropy*, i per tant ha estat una bona idea crear models separats pel volum central i pels plans, i calcular l'*excess entropy* només quan cal.

Per acabar amb l'anàlisi dels temps, comentarem el temps de resposta en interaccionar amb un model. En tots els casos veiem que aquest temps de resposta és força petit, normalment menys d'un segon. Això ha estat gràcies a les classes vtkInteractorStyle*Ggg, ja que amb les originals de VTK trigava alguns segons en tots els casos. El temps de resposta per interaccionar amb la càmera és proper a 0 en tots els casos.

Les funcions de transferència definides en les proves dels Miralls Màgics són tant bones i realistes com hem pogut aconseguir. Trobar una bona funció de transferència és una feina complicada i cal tenir paciència per anar fent proves, ja que no hi ha cap mètode que ho faciliti.

Les funcions de transferència de les proves de la selecció del punt de vista òptim són les s'han creat automàticament després de la segmentació. Amb el model sintètic no importen gaire els colors, però amb l'altre no ha quedat realista. No obstant, ja hem demostrat a les proves dels Miralls Màgics que amb paciència es poden aconseguir uns colors més realistes, i en aquest cas els colors assignats permetien diferenciar fàcilment els clústers.

Acabarem comentant que el model sintètic ens ha servit per comprovar que l'algorisme de segmentació funciona, ja ha encertat tots els límits, fins i tot separant el cub més petit, que gairebé no es veu.

8 Millores i ampliacions

Tot i que les tècniques i mètodes que hem explicat i implementat ens han permès assolir els objectius mínims que ens havíem fixat, som conscients que moltes d'aquestes tècniques poden ser millorades. En aquest capítol recollim de forma molt sintetitzada algunes d'aquestes possibles millores i en alguns casos com es podrien aplicar.

- **Implementar un sistema de prioritats de visualització en els Miralls Màgics per a models fusionats.**

  No seria gaire difícil d'implementar aprofitant la característica del vtkRenderer que dibuixa els objectes en l'ordre en què li han estat afegits, tal com hem comentat al capítol d'implementació. Caldria dissenyar una interfícies que permetés definir aquestes prioritats i per aplicar-les només caldria treure els objectes del *renderer* i afegir-los seguint un ordre creixent de prioritat.

- **Actualitzar els miralls sense obrir una finestra de visualització.**

  Amb la implementació actual és impossible. Caldria buscar una altra manera d'actualitzar els miralls. També existeix una altra solució que es podria aplicar sense fer gaires canvis al codi, però requeriria una instal·lació especial i complicada de les VTK i Mesa. Això s'anomena VTK amb "Mangled Mesa" i permet fer visualitzacions en una mena de "finestra virtual". Aquesta opció l'hem descartat perquè l'aplicació ha de poder executar-se en entorns més estàndards.

- **Actualitzar en temps real els miralls dels Miralls Màgics.**

  És difícil de fer perquè l'actualització dels miralls és molt costosa. Només seria pràctic si no apareguessin finestres de visualització per actualitzar els miralls.

- **Obrir finestres de visualització en una altra pantalla.**

  En el mètode de selecció del punt de vista òptim hi ha una finestra de visualització per cada pla i és molt difícil veure-les totes alhora perquè ocupen una part molt gran de la pantalla, limitant la visualització principal. La solució per veure la finestra principal i les dels plans alhora seria fer-ho amb més d'una pantalla. A la primera pantalla hi hauria la finestra principal i a l'altra o les altres hi hauria les finestres dels plans. Caldria mirar bé la documentació de les Qt i les VTK per veure si això es pot fer i com.

- **Calcular l'*excess entropy* del volum central.**

  Actualment només calculem l'*excess entropy* dels plans, però seria interessant calcular-lo també per al volum central, és a dir, des del punt de vista de la càmera principal. Això no seria gaire complicat de fer. Caldria utilitzar les mateixes classes de visualització que pels plans, que permeten fer el càlcul, i fer alguna adaptació més, decidint a quina classe encarreguem la feina de calcular l'*excess entropy* del volum central, ja que per aquest no tenim cap pla que se'n pugui encarregar.

- **Estudiar altres maneres de presentar els resultats del càlcul de l'*excess entropy*.**

  Potser seria més útil que els resultats apareguessin al costat del pla, o impresos directament sobre el pla. Una altra possibilitat seria obrir un finestra on es mostressin tots els plans amb els valors calculats.

- **Estudiar altres mesures de la teoria de la informació.**

  Caldria fer un estudi exhaustiu d'altres mesures de la Teoria de la Informació per analitzar el seu comportament. També seria bo estudiar diferents modalitats d'imatge per poder determinar quina és la millor mesura.

- **Mostrar els valors dels punts de la funció de transferència.**

  Això permetria definir funcions de transferència amb més exactitud. Per exemple es podria mostrar un *tooltip* amb les coordenades quan el ratolí passi per sobre d'un punt de la funció.

- **Desar i carregar funcions de transferència.**

  Seria útil poder guardar una funció de transferència en un fitxer per poder-la carregar i utilitzar més tard. Per exemple, si l'usuari troba una bona funció de transferència amb l'ajuda del model segmentat, podria interessar-li fer servir aquella mateixa funció de transferència amb els Miralls Màgics. Les funcions de transferència podrien guardar-se amb un format XML, aprofitant que les Qt donen facilitats per treballar amb aquest llenguatge.

- **Millorar la interfície d'usuari afegint *tooltips*, ajudes contextuals, dreceres del teclat, etc.**

  Ja hi ha alguns *tooltips* i dreceres del teclat. Caldria fer-ho per a tots els elements de la interfície que ho puguin necessitar. També caldria tenir en compte la navegació amb el teclat, mitjançant la tecla de tabulació.

- **Traduir l'aplicació a altres idiomes.**

  Això és fàcil de fer amb el Qt Linguist. Només requereix una mica de temps de dedicació.

- **Millorar el rendiment de l'aplicació.**

  Hem procurat fer un codi eficient en els punts clau, però caldria mirar els talls de codi que s'executen més sovint per veure si se'n pot millorar més el rendiment. Això seria especialment important en els trossos de codi que s'executen durant un *ray casting*, com és el cas del càlcul de l'*excess entropy*.

- **Tenir en compte els suggeriments dels usuaris finals.**

  Els usuaris finals són els que poden tenir més idees per proposar millores.

9 Conclusions

Els objectius generals d'aquest projecte eren *implementar la tècnica dels Miralls Màgics per facilitar la visualització i interpretació de models de vòxels simples i fusionats i implementar un algorisme que ajudés a determinar els millors punts de vista per visualitzar un model de vòxels, i integrar-los tots dos a una plataforma de tractament d'imatges mèdiques*.

Hem assolit tots els objectius definits, concretament:

- Hem implementat la tècnica bàsica dels Miralls Màgics, que consisteix a fer la visualització d'un model de vòxels al centre de la pantalla i situar al seu voltant uns "miralls" on hi ha la visualització del model des del punt de vista del mirall.

- Hem estès la tècnica dels Miralls Màgics per tal de suportar models fusionats, permetent visualitzar diferents propietats a cada mirall.

- L'usuari pot triar el nombre de miralls i definir la posició de cadascun.

- L'usuari pot configurar tots els paràmetres de visualització per cada mirall i propietat, incloent la funció de transferència i la visibilitat o no visibilitat. A més a més hem deixat oberta la possibilitat d'aplicar un ombrejat o no.

- Totes les opcions són accessibles a través d'una interfície gràfica integrada a la plataforma.

- Hem implementat una tècnica que ajuda a trobar els millors punt de vista per a un model. Aquesta tècnica es basa en el càlcul de l'*excess entropy*, un concepte definit a la Teoria de la Informació que mesura la quantitat d'informació.

A més a més hem aconseguit una eficiència prou bona amb l'ajuda de les biblioteques utilitzades i fent algunes modificacions a les classes. El resultat final és una aplicació amb una bona interactivitat. Tot i això encara és millorable.

La modificació d'una classe de VTK i el mecanisme de *signals i slots* de Qt, ens han permès implementar fàcilment el càlcul de l'*excess entropy*.

L'usuari pot manipular directament la visualització amb el teclat i el ratolí, gràcies a Qt i VTK.

L'aplicació calcula automàticament la mida dels miralls i els plans perquè s'hi pugui visualitzar completament el model tingui la rotació que tingui.

Les interfícies gràfiques permeten configurar totes les opcions de cada mètode: les opcions de visualització generals, per cada mirall i per cada volum en el cas dels Miralls Màgics; els paràmetres de segmentació i els de visualització en el cas de la selecció del punt de vista. També hem procurat que la interfície fos intuïtiva i fàcil de fer servir. Les interfícies han estat dissenyades amb Qt i el Qt Designer.

El mètode de selecció del punt de vista segmenta el model i defineix una funció de transferència automàtica que s'ajusta al model segmentat, definint un color diferent per cada clúster.

Hem aconseguit un disseny modular, cohesionat i poc acoblat.

Hem implementat tot el projecte fent servir eines de domini públic i lliures. Per implementar hem fet servir les eines comentades a l'apartat d'objectius. Aquesta memòria ha estat redactada amb l'OpenOffice.org Writer 2.0.2. Els diagrames han estat fets amb l'ArgoUML 0.22. Algunes edicions d'imatges han estat fetes amb El GIMP 2.2.12. El diagrama de planificació ha estat fet el KPlato 0.5.2, que forma part del KOffice.

Per acabar, tot i que els objectius han estat assolits sempre és possible fer millores a l'aplicació perquè ofereixi més possibilitats i sigui més eficient i útil. També hagués calgut fer més proves amb els mateixos models i també amb models diferents per comprovar el rendiment i trobar possibles errors.

9.1 Conclusions personals

La realització d'aquest projecte m'ha permès aprendre noves coses sobre algunes tecnologies com les Qt i les VTK. Ja coneixia el funcionament de les Qt 3, però amb les Qt 4 i ha molts canvis i m'ha calgut aprendre noves maneres de fer algunes coses. Pel que fa a les VTK, també en coneixia la versió 4.5.0, i en aquest cas no hi ha gaires canvis a la versió 5.0. També he après algunes coses que no sabia de C++. Una tecnologia que he après completament nova és Subversion, per fer el control de versions.

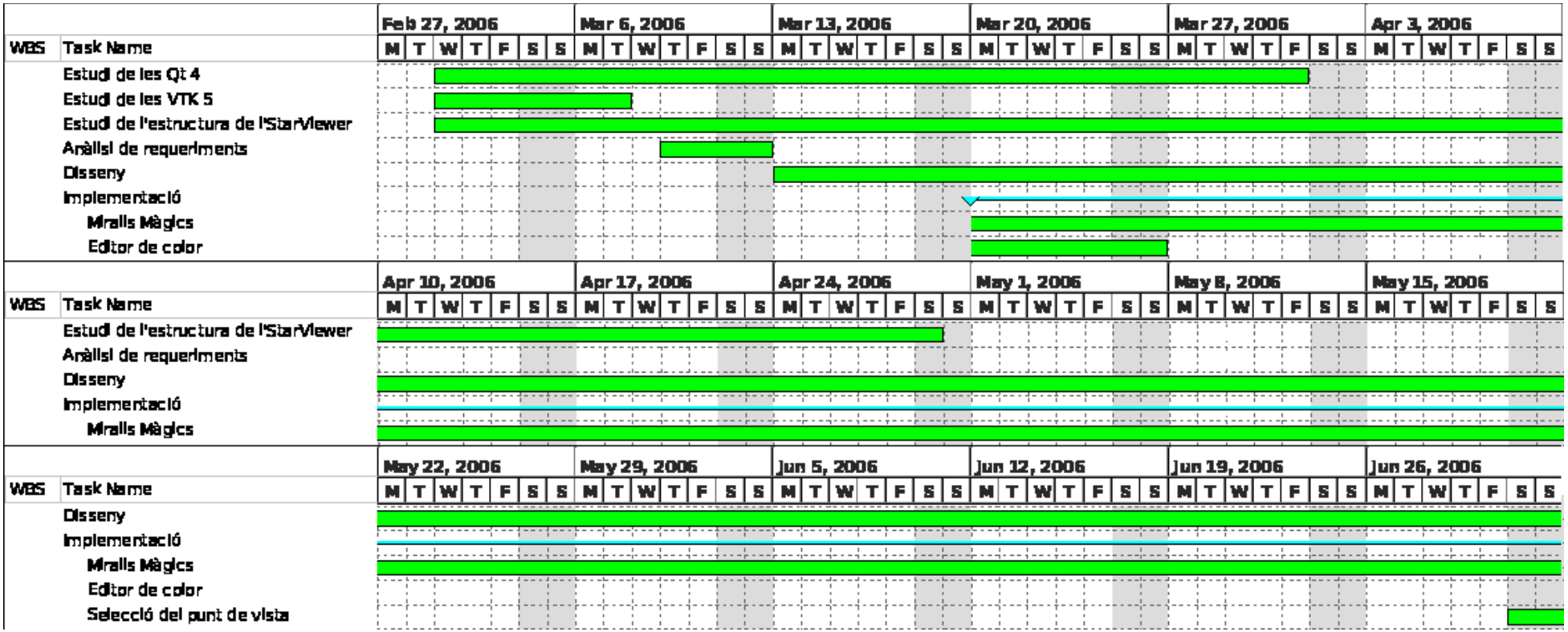
A banda de les noves tecnologies, també vull destacar que he pogut veure com funcionava un algorisme de segmentació d'un model de vòxels. Ja havia vist algorismes de segmentació d'imatges 2D, però són molt diferents del mètode que he implementat aquí.

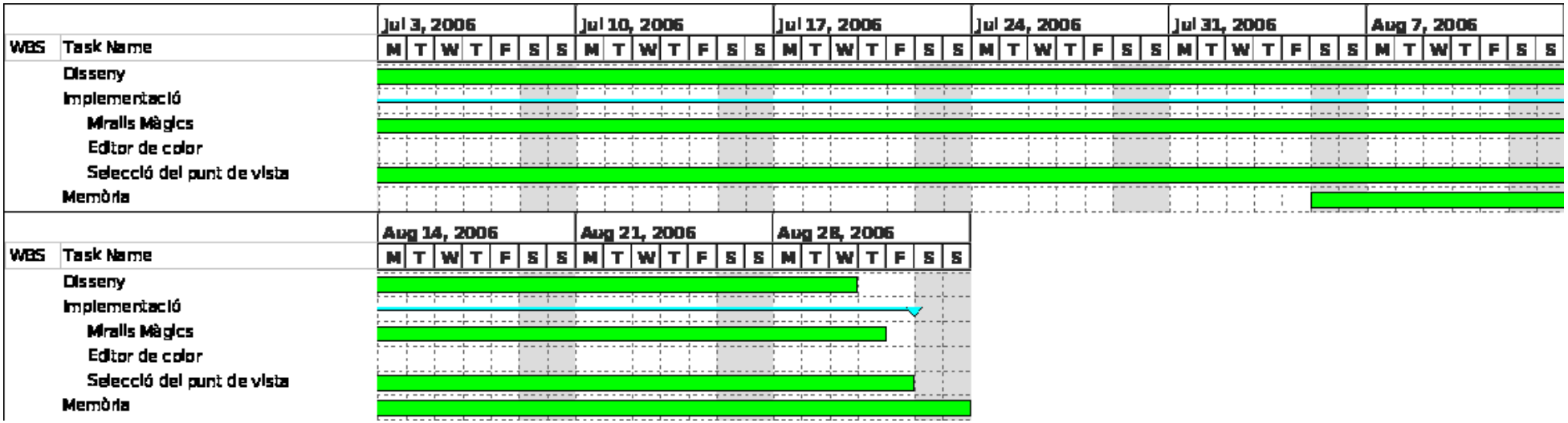
Un altre aspecte remarcable és que he hagut d'aplicar diverses tècniques per aconseguir una bona optimització en temps en els punts més crítics.

També he pogut comprovar i aplicar els beneficis d'aprofitar el codi existent. Hi ha diferents mòduls en el meu codi que són molt semblants, i molts cops he pogut aprofitar les millores que feia en un lloc per aplicar-les a un altre lloc amb pocs canvis. Hi ha algunes classes que són modificacions de les que tenen les eines lliures que he fet servir. L'editor de la funció de transferència està basat en una *demo* de les Qt 4 per demostrar les noves possibilitats de dibuixar gradients. La part d'agafar mostres del model de vòxels per calcular l'*excess entropy* l'he fet fàcilment modificant una classe de VTK encarregada del *ray casting*.

Per acabar, vull dir que estic satisfet i content amb els resultats aconseguits i els nous coneixements apresos, si bé m'hagués agradat tenir més temps per donar-li un acabat millor a l'aplicació.

9.2 Planificació seguida


Feb 27, 2006
Mar 6, 2006
Mar 13, 2006
Mar 20, 2006
Mar 27, 2006
Apr 3, 2006
M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S
WBS
Task Name
Estudi de les Qt 4
Estudi de les VTK 5
Estudi de l'estructura de l'StarViewer
Anàlisi de requeriments
Disseny
Implementació
Miralls Màgics
Editor de color
Apr 10, 2006
Apr 17, 2006
Apr 24, 2006
May 1, 2006
May 8, 2006
May 15, 2006
M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S
WBS
Task Name
Estudi de l'estructura de l'StarViewer
Anàlisi de requeriments
Disseny
Implementació
Miralls Màgics
May 22, 2006
May 29, 2006
Jun 5, 2006
Jun 12, 2006
Jun 19, 2006
Jun 26, 2006
M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S
WBS
Task Name
Disseny
Implementació
Miralls Màgics
Editor de color
Selecció del punt de vista


Jul 3, 2006
Jul 10, 2006
Jul 17, 2006
Jul 24, 2006
Jul 31, 2006
Aug 7, 2006
M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S
WBS
Task Name
Disseny
Implementació
Miralls Màgics
Editor de color
Selecció del punt de vista
Memòria
Aug 14, 2006
Aug 21, 2006
Aug 28, 2006
M T W T F S S M T W T F S S M T W T F S S
WBS
Task Name
Disseny
Implementació
Miralls Màgics
Editor de color
Selecció del punt de vista
Memòria

Referències

[1] Object Management Group, Inc.. **Object Management Group - UML** - http://www.uml.org/

[2] Wikipedia, the free encyclopedia. **Unified Modeling Language** - http://en.wikipedia.org/wiki/Unified_Modeling_Language

[3] Viquipèdia, l'enciclopèdia lliure. **Programació Extrema** - http://ca.wikipedia.org/wiki/Programaci%C3%B3_Extrema

[4]Anonymous. **Medical Image Segmentation using Excess Entropy**.

[5] Ramon Mas Sansó. **Tema IV: Els models d'il·luminació global**. 2004 - http://dmi.uib.es/~ramon/Docencia/ig2/IG2TemaIVe.pdf

[6] Andreas König, Helmut Doleisch, and Eduard Gröller. **Multiple Views and Magic Mirrors – fMRI Visualization of the Human Brain**. 2001 - http://www.cg.tuwien.ac.at/research/vis/vismed/MM/

[7] Melanie Tory. **Mental Registration of 2D and 3D Visualizations (An Empirical Study)**. 2003 - http://www.cs.uvic.ca/~mtory/publications/vis03.pdf

[8] Rob Womersley. **Distributing points on the sphere** - http://www.maths.unsw.edu.au/school/articles/me100.html

[9] Trolltech. **Qt Homepage** - http://www.trolltech.com/products/qt

[10] Insight Software Consortium. **Insight Segmentation and Registration Toolkit** - http://www.itk.org/

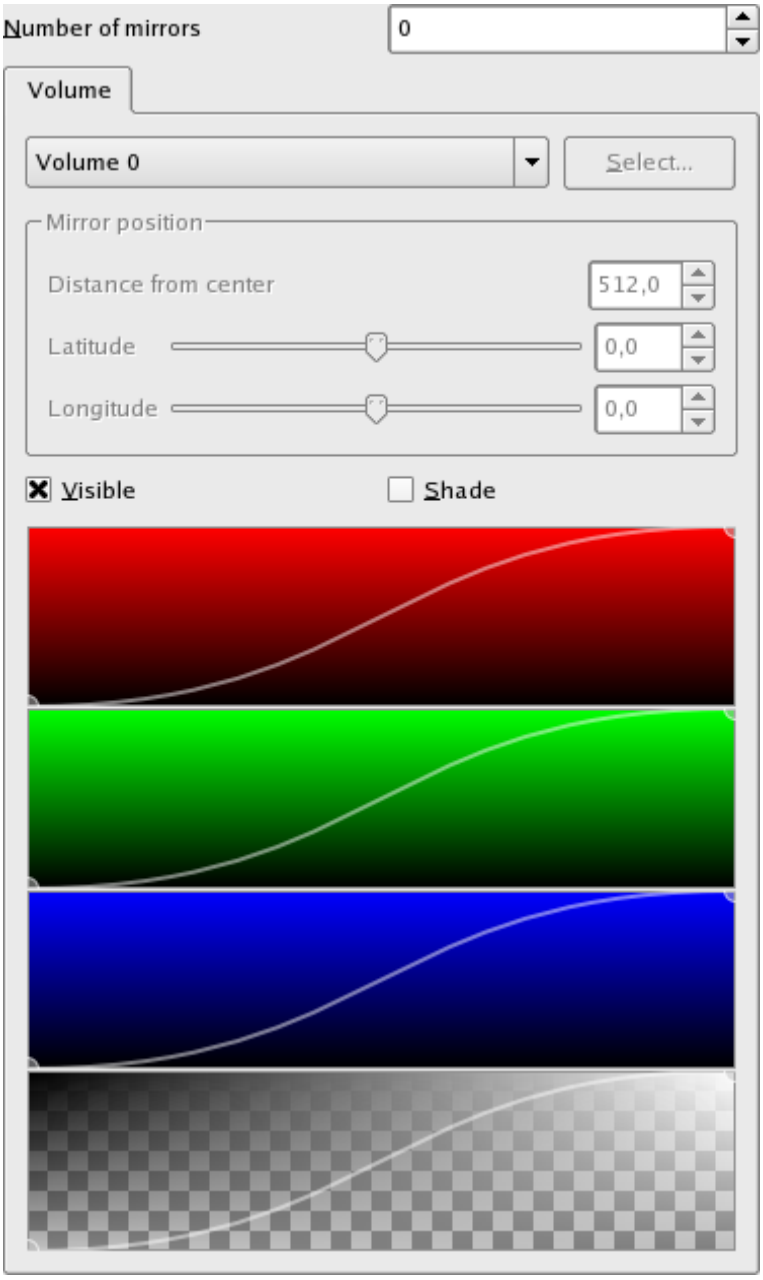
[11] Kitware. **VTK Home Page** - http://www.vtk.org/

Manual d’usuari

En aquest manual d’usuari partim de la base que l’usuari ja ha obert un o més models amb la plataforma.

Miralls Màgics

Per accedir a aquesta funcionalitat cal anar a la secció dels algorismes de visualització i seleccionar-la. Llavors trobarem un una interfície com la següent:



És la interfície per configurar els paràmetres dels Miralls Màgics. El primer que cal fer és seleccionar el/s volum/s que vulguem visualitzar prement el botó "Select...", seleccionant-los al quadre de diàleg que apareixerà i prement el botó d'acceptar.

El següent pas és triar el nombre de miralls mitjançant l'*spin box* destinat a aquesta tasca. Llavors per cada mirall, accessibles des de les pestanyes, es poden realitzar les tasques següents:

- Definir la posició del mirall mitjançant els controls que ho indiquen. Concretament, cal definir la distància des del centre de l'espai, la latitud i la longitud. Aquests dos últims són anàlegs a la latitud i la longitud d'un punt de la Terra. No es pot definir la posició del volum central.

- Triar el volum que vulguem configurar, si n'hi ha més d'un.

- Per cada volum es poden realitzar les tasques següents:

  - Definir si és visible o no en el mirall concret que estiguem configurant.

  - Triar si volum que es pinti amb ombrejat o no al mirall concret que estiguem configurant.

  - Definir-ne la funció de transferència per al mirall concret que estiguem configurant. La funció de transferència es defineix mitjançant punts. Per crear un punt s'ha de fer clic sobre un espai buit. Per moure un punt cal arrossegar-lo. Per esborrar un punt cal fer clic amb el botó dret a sobre seu.

Un cop definits tots els paràmetres per tots els miralls i tots els volums, només cal prémer el botó "Apply" per obtenir la primera visualització.

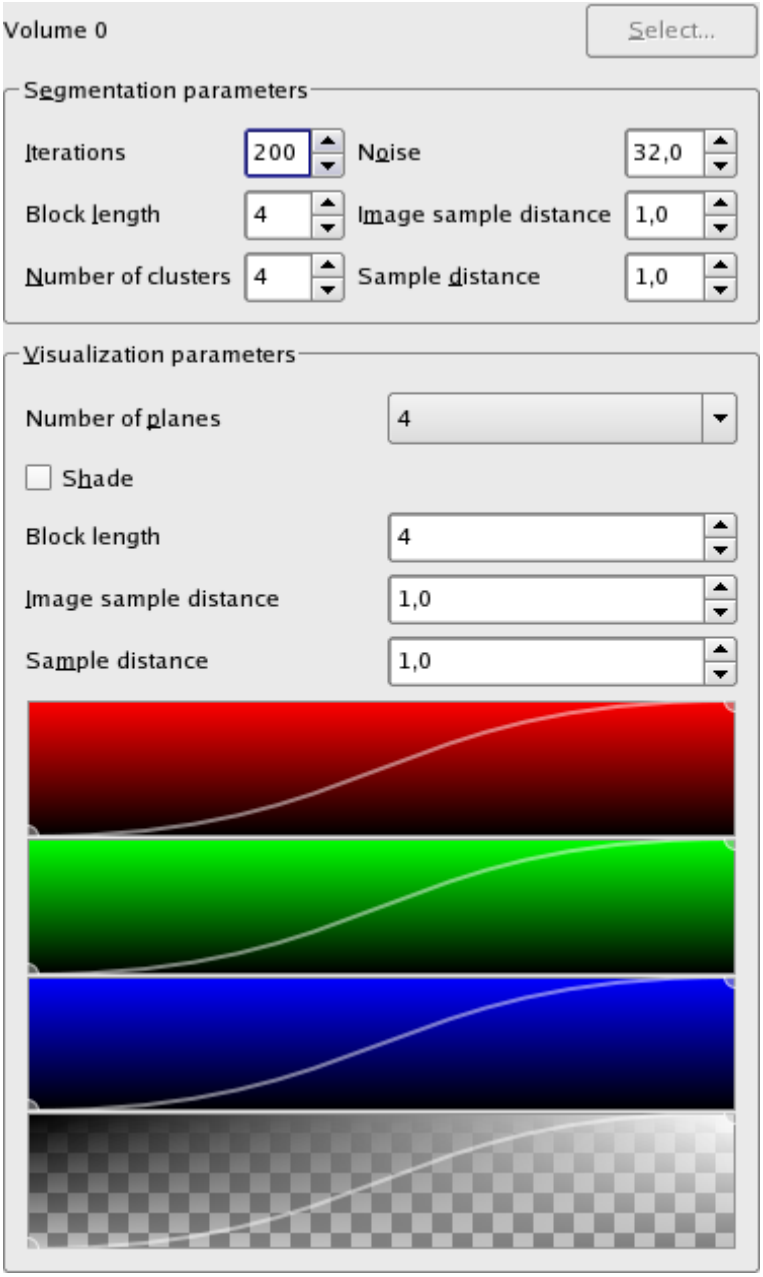
Ara podem interaccionar amb aquesta visualització utilitzant el ratolí, i també el teclat. Amb la configuració inicial el ratolí mou la càmera, però prement la tecla 'A' podem moure el model, i prement la tecla 'C' tornem a controlar la càmera. Arrossegant el ratolí mantenint premut el botó esquerre podem girar la càmera al voltant del model o girar el model. Si arrosseguem amb el botó del mig o la roda es desplaça la càmera o el model. Amb el botó dret es fa zoom o s'escala el model. Girant la rodeta també podem fer zoom si estem controlant la càmera, però no té cap efecte sobre el model.. A més a més hi ha dos estils d'interacció amb cada element: *joystick* i *trackball*. Al primer (predeterminat) s'hi accedeix prement la tecla 'J', i al segon amb la tecla 'T'. També podem prémer la tecla 'R' per posar la càmera a una distància que permeti veure el volum i tots els miralls.

Si s'interacciona amb el model els miralls s'actualitzen automàticament.

També es poden canviar els paràmetres de visualització i aplicar els nous per actualitzar la visualització.

Selecció del punt de vista òptim

Per accedir a aquesta funcionalitat cal anar a la secció dels algorismes de visualització i seleccionar-la. Llavors trobarem un una interfície com la següent:



És la interfície per configurar els paràmetres de la selecció del punt de vista òptim. El primer que cal fer és seleccionar el volum que vulguem visualitzar prement el botó "Select...", seleccionant-lo al quadre de diàleg que apareixerà i prement el botó d'acceptar.

El següent pas és definir els paràmetres de la segmentació, que són els següents:

- Nombre d'iteracions de l'algorisme genètic.
- Longitud de bloc per calcular l'*excess entropy*.
- Nombre de clústers.
- Soroll per l'algorisme genètic.

- Distància entre raigs.
- Distància entre mostres.

Després de definir els paràmetres de la segmentació es poden definir els de la visualització o bé prémer el botó "Apply" per fer la segmentació i la primera visualització amb els paràmetres per defecte. Els paràmetres de visualització són els següents:

- Nombre de plans a situar al voltant del model.
- Si volem ombrejat o no.
- Longitud de bloc per calcular l'*excess entropy*.
- Distància entre raigs.
- Distància entre mostres.
- Funció de transferència. Es defineix com en el cas dels Miralls Màgics.

La segmentació triga una estona, i quan apareix la visualització podem interaccionar amb ella de la mateixa manera que amb els miralls màgics. També s'haurà definit una funció de transferència automàtica ajustada al model segmentat.

Quan es calculi l'*excess entropy* apareixerà una finestra de resultats com la següent:

En teoria el millor pla és el que té l'*excess entropy* més alta.

Podem canviar els paràmetres de la visualització tantes vegades com vulguem i aplicar-los. Quan calgui es calcularà l'*excess entropy*.